Formalismes CFTL pour les arbres sans rang

Adrien BOIRET

August 19, 2010

Afin de décrire de manière finie des langages infinis de mots, plusieurs formalismes ont été développés. Parmi eux, les expressions régulières fournissent d'intéressantes propriétés, comme la stabilité par union, intersection, complément, ainsi que des problèmes de décision (par ex. vide et appartenance). De plus, le théorème de Kleene montre qu'il existe un formalisme d'automates qui reconnait la même classe de langage que ces expressions régulières, c'est-à-dire qu'à partir d'une expression régulière on peut "compiler" un automate qui reconnait exactement les mots compris dans le langage décrit par l'expression.

Cependant, il a fallu considérer un formalisme plus expressif afin de décrire des langages que les expressions régulières sont dans l'incapacité de représenter, comme par exemple l'ensemble des mots bien parenthésés, ou plus généralement les codes d'un langage de programmation. Les grammaires Context-Free répondent à ce besoin, et sont, elles aussi, équivalentes à un formalisme d'automates (les automates à pile), c'est-à-dire que tout langage décrit par une grammaire Context-Free est reconnu par un automate à pile. Quand bien même ce gain d'expressivité est acquis au dépend de la stabilité par complément et intersection, ces formalismes équivalents sont un outil précieux dans le domaine de la compilation d'un langage de programmation décrit par une grammaire Context-Free, comme en témoignent les compilateurs de compilateurs comme YACC.

L'intérêt porté aux langages d'arbres vient du désir de traiter des problèmes qui ne sont pas séquentiels par nature, comme l'accès à une partie bien précise d'un fichier HTML. Le standard XML permet de décrire ces fichiers comme des arbres plutôt que comme une liste de lignes, ce qui facilite la lecture, à l'instar d'un lecteur qui arriverait directement à la page désirée d son livre, non pas en ayant à lire tout le livre jusque là, mais en se repérant grâce à l'index.

On est amené naturellement a développer pour les langages d'arbres des outils de description et de reconnaissance comme ceux que l'on a développés pour les langages de mots. Les formalismes d'automates d'arbres "de base" sont décrits dans [TATA], et reconnaissent, par exemple, le langage des fichiers bien formés en XML.

Cependant, comme pour les mots, ces formalismes peuvent être insuffisants

pour décrire certains langages; par exemple, ce papier est motivé par le besoin de trouver une classe de langages stable pour la création de vues de fichiers XML.

Une idée en ce sens consiste à définir, sur le même modèle que celles des mots, des grammaires Context-Free d'arbres. [Guessarian] décrit un formalisme intéressant de langages d'arbres context-free (Context Free Tree Langage, ou CFTL) pour les arbres de rang borné: plus expressif que les formalismes d'automates d'arbres "de base" de [TATA], il est également équivalent à un formalisme d'automates d'arbres à pile, ce qui permet de reconnaitre efficacement les langages décrits.

Cependant, ce formalisme ne s'étend pas au cas général des arbres sans rang, qui sont nécessaires pour travailler sur des fichiers XML, sans rang par nature. La seule manière de se ramener à ce formalisme est d'utiliser un codage vers les arbres binaires, comme First Child Next Sibling (FCNS) , aux arbres sans rang afin de leur imposer un rang borné.

Le but de ce papier est de trouver un formalisme CFTL pour les arbres sans rang qui étendrait celui de [Guessarian] tout en étant équivalent à un formalisme d'automates. Il doit également être plus puissant que CFTL rang borné combiné à l'encodage FCNS.

La difficulté vient de la perte de l'information que fournissait le rang sur le nombre de fils de chaque noeud, et de l'impératif de devoir les manipuler assez finement pour être au moins aussi expressif que CFTL rang borné. Le problème est le même pour les automates, car l'absence d'un nombre fixe de fils empêche d'adapter directement la méthode employée par [Guessarian], qui est de définir une règle par fils lors de la descente de la tête de lecture.

On désire les propriétés suivantes pour notre formalisme:

- Il doit étendre les CFTL rang borné, c'est-à-dire décrire au moins tous les langages CFTL rang borné, mais aussi tous les langages de $FCNS^{-1}$(CFTL rang borné), c'est-à-dire tous les langages L dont l'encodage FCNS(L) est un langage CFTL rang borné.
- Les problèmes du vide et de l'appartenance doivent être décidable: entre autres, si le problème d'appartenance n'est pas décidable, aucun formalisme d'automates ne sera équivalent.
- On doit trouver un formalisme d'automates d'arbres équivalent aux grammaires, afin de reconnaitre les langages décrits.

Ce rapport ne présente pas de formalisme remplissant les deux premières conditions, ce qui explique que le troisième point reste à résoudre.
On étudiera deux formalismes:

- Le premier (formalisme Unary), qui manque de l'expressivité nécessaire à décrire CFTL rang borné, car il n'a accès qu'à tous ses fils en même temps.

- Le second (formalisme Head:Tail), est assez expressif mais ses problèmes du vide et d'appartenance sont indécidables. Ici, le premier fils est accessible indépendamment des autres.

Pour finir, comme aucun des deux formalismes ne convient, on considèrera une restriction de Head:Tail où l'on force la dérivation à se faire dans un sens bien précis, de haut en bas (Top-Down). On observera les résultats conservés et les nouveaux.

Contents

Part I
Langages d'arbres Context-Free rang borné

Les context-free tree languages (CFTL) pour les arbres de rang borné ont déjà un formalisme très robuste, équivalent à un formalisme d'automates sur les arbres de rang borné, Push-Down Tree Automata (PDTA) et PDTA restreint, comme démontré dans [Guessarian]. Le but de ce papier aurait été de trouver un formalisme aux propriétés similaires généralisés aux arbres sans rang, afin de développer des outils de vérifications sur les arbres sans rang.

1 Définitions

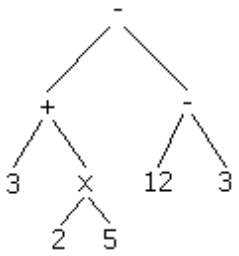
L'ensemble des arbres de rang borné sur un alphabet à rang borné $\mathcal{F}$ est l'ensemble des termes bien formés sur cet alphabet

Définition. *Un alphabet à rang borné $\mathcal{F}$ est un ensemble fini de symboles f munis chacun d'une arité (ou rang) a_f qui correspond au nombre de fils qu'il attend.*
L'ensemble des arbres de rang borné sur un alphabet $\mathcal{T}(\mathcal{F})$ est le plus petit ensemble tel que:

- $\epsilon \in \mathcal{T}(\mathcal{F})$, *c'est l'arbre vide.*
- *Si* $f \in \mathcal{F}$ *et* $a_f = 0$, $f \in \mathcal{T}(\mathcal{F})$.
- *Si* $f \in \mathcal{F}$, $a_f = k$ *et* $T_1...T_k \in \mathcal{T}(\mathcal{F})$ *sont* k *arbres non-vides, alors* $f(T_1...T_k) \in \mathcal{T}(\mathcal{F})$.

Chaque symbole de $\mathcal{F}$ dans un arbre est un noeud de cet arbre.
Dans les 2 derniers cas, f est la racine.
Dans le dernier cas, f est l'ancêtre des noeuds de $T_1...T_k$, le père des racines de $T_1...T_k$. Ces racines sont les fils de f.
Un noeud sans fils est une feuille.

Exemple. *L'arithmétique sur les entiers:*
Les entiers sont d'arité 0, les opérateurs d'arité 2.
$(3 + (2 \times 5)) - (12 - 3)$ *est représenté par l'arbre:*
Exemple:



Ces arbres de rang borné sont dotés de divers formalismes qui permettent de décrire ou reconnaitre des langages, comme les automates d'arbres (voir [TATA]). On parlera ici du formalisme CFTL, qui décrit des langages à l'aide d'un grammaire context-free, à la manière d'une grammaire sur les mots.

2 Grammaires CFTL rang borné

Une grammaire CFTL rang borné fonctionne comme une grammaire sur les mots: on a des non-terminaux N, dont un initial S_0, et des règles qui les dérivent. La différence est qu'au lieu de mots, chaque dérivation crée un arbre, qui doit gérer les fils du noeud dérivé au lieu de la simple suite du mot.

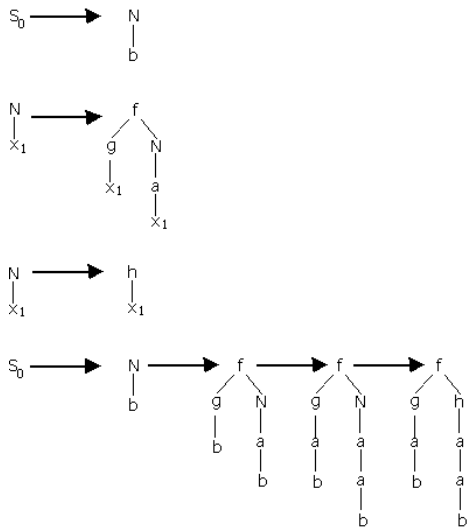
Définition. *Une grammaire CFTL rang borné G sur $\mathcal{F}$ est la donnée de:*

- *Un alphabet rang borné $\mathcal{N}$ de symboles non-terminaux.*
- *Un symbole initial $S_0 \in \mathcal{N}$ d'arité 0.*
- *Un ensemble de règles Δ.*

Une règle de Δ est de la forme $N(x_1...x_k) \to T(x_1...x_k)$ où $a_N = k$ et $T(x_1...x_k)$ est un arbre sur $\mathcal{T}(\mathcal{F} \cup \mathcal{N} \cup \{x_1...x_k\})$, où $x_1...x_k$ sont d'arité 0 (ne peuvent être que des feuilles).

Une dérivation selon cette règle se déroule comme suit: le noeud dérivé N est remplacé par $T(x_1...x_k)$ où chaque occurrence de x_i est respectivement remplacée par le i^{eme} fils de N.
Un arbre de $\mathcal{T}(\mathcal{F})$ est reconnu par G si on l'obtient après une ou plusieurs dérivations selon une règle de Δ depuis S_0.
Le langage $L(G)$ est l'ensemble des arbres reconnus par G.

Exemple.

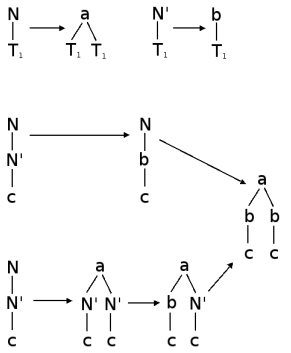


On démontrera que sur ces grammaires, on peut dériver en premier les noeuds les plus hauts (dérivation Top-Down).

Lemme I.1. *On peut décider de ne dériver que les non-terminaux les plus hauts, i.e des non-terminaux qui n'ont aucun ancêtre non-terminaux, sans modifier le langage reconnu par une grammaire.*

Démonstration. Cette contrainte ne crée aucune nouvelle dérivation, aussi le langage obtenu est un sous-ensemble du langage original, ou égal au langage original.
Afin de rendre une dérivation top-down, on choisit un non-terminal le plus haut N. Sa dérivation $N(x_1...x_k) \to T(x_1...x_k)$ est précédée de la dérivation de certains de ses descendants, c'est-à-dire:
$N(T_1...T_k) \to N(T'_1...T'_k) \to T(T'_1...T'_k)$, avec
$T_1...T_k \to T'_1...T'_k$ la transformation des descendants de N.
On peut pour rétablir l'ordre top-down effectuer $N(x_1...x_k) \to T(x_1...x_k)$ en premier, puis appliquer les dérivations menant de T_1 resp. $T_2...T_k$ à T'_1 resp. $T_2...T'_k$ à chaque occurrence de T'_1 resp. $T_2...T'_k$ créée par la dérivation de N.
On répète ce processus jusqu'à ce que la dérivation soit top-down.
Afin de se convaincre de la terminaison de ce processus, on peut considérer le multi-ensemble des dérivations de la liste de départ, ordonnée par priorité top-down, à savoir, si une dérivation doit être effectuée avant une autre (si elle crée le non-terminal dérivé par cette autre), elle est plus "grande". Si une dérivation a lieu sous une autre, elle est plus petite.
Chaque étape du processus multiplie, au pire, par un nombre fini (nombre de X dans la règle de Δ) le nombre des dérivations plus petites, et peut se débarrasser de la dérivation triée. On fait donc décroitre le multi-ensemble, le processus se termine donc.

Exemple.



L'avantage majeur de ce formalisme est qu'il est équivalent à une certaine classe d'automates: les automates d'arbres à pile Push-Down.

Un PDTA (Push-Down Tree Automata) est un automate d'arbre à pile, mais dont la pile a une structure d'arbre. Il peut lire son état actuel, l'étiquette du noeud sur lequel il se trouve, et la racine de son arbre-pile.

Définition. *Un Push-Down Tree Automaton (PDTA) $\mathcal{A}$ sur un alphabet à rang borné $\mathcal{F}$ est la donnée de:*

- *Q un ensemble fini d'états.*
- *Π un alphabet de pile: c'est un alphabet fini de rang borné utilisé pour construire l'arbre-pile.*
- *$q_0 \in Q$ un état initial.*
- *$Z_0 \in \Pi$ le symbole initial, d'arité 0. Il s'agit de la pile initiale.*
- *R un ensemble de règles.*

R est constituée de règles de deux sortes:

- *Règles de lecture:*
  $q(f(v_1...v_n), E(x_1...x_m)) \rightarrow f(q_1(v_1, \pi_1), ..., q_n(v_n, \pi_n))$

  où $f \in \mathcal{F}$ d'arité n, $E \in \Pi$ d'arité m,
  $\pi_1...\pi_n \in \mathcal{T}(\Pi \cup \{x_1...x_m\})$ où $x_1...x_m$ d'arité 0 et ne peuvent donc être que des feuilles,
  $q, q_1...q_n \in Q$.
- *Règles ϵ:*
  $q(v, E(x_1...x_m)) \rightarrow q'(v, \pi')$

  où $E \in \Pi$ d'arité m,
  $\pi' \in \mathcal{T}(\Pi \cup \{x_1...x_m\})$ où $x_1...x_m$ d'arité 0 et ne peuvent donc être que des feuilles,
  $q, q' \in Q$.

Les règles de lecture enclenche la lecture de l'étiquette du noeud où se trouve l'automate et du sommet de la pile, puis déplace la tête de lecture aux fils du noeud actuel avec en arbre-pile une transformation de la pile actuelle, privée de sa racine (ce qu'on peut traduire comme un Pop).
Les règles ϵ ne nécessitent pas de lecture et permettent une transformation statique de la pile ainsi qu'un changement d'état.

- *Règles de lecture: si la tête de lecture est sur la racine de $f(T_1...T_n)$, à l'état q, et que la pile est de la forme $E(\pi'_1...\pi'_m)$ on peut utiliser:*
  $q(f(v_1...v_n), E(x_1...x_m)) \rightarrow f(q_1(v_1, \pi_1), ..., (q_n(v_n, \pi_n))$

  La tête de lecture sera alors transmise à chaque T_i, dans l'état q_i, avec la pile $\pi_i(\pi'_1...\pi'_m)$, c'est-à-dire π_i où chaque instance de x_k est remplacée par π'_k.

- *Règles ϵ: si la tête de lecture est dans l'état q, et que la pile est de la forme $E(\pi'_1...\pi'_m)$, alors quelque soit le noeud sur lequel on se trouve, on peut appliquer:*
  $q(v, E(x_1...x_m)) \rightarrow q'(v, \pi')$

  La tête de lecture restera immobile, passera de l'état q à l'état q', et sa pile deviendra $\pi'(\pi'_1...\pi'_m)$, c'est-à-dire π' où chaque instance de x_k est remplacée par π'_k. Un arbre T est accepté si l'on peut utiliser un chemin de règles tel que:
  $q_0(T, Z_0) \rightarrow T$, *c'est-à-dire que l'on pousse la tête de lecture de la racine jusqu'aux feuilles.*

On peut décider de reconnaitre un langage par état final et/ou par pile vide (ces deux formalismes sont équivalents, comme pour les automates sur les mots). [Guessarian] étudie plusieurs variantes sur les définitions, la reconnaissance, des généralisations et des restrictions. Dans le cadre de notre démarche, la reconnaissance par état final suffira.

Ce formalisme PDTA est équivalent au CFTL rang borné:

Théorème I.1. *Un langage d'arbre rang borné est décrit par une grammaire Context-Free si et seulement si il est reconnu par un PDTA.*

Démonstration. Voir [Guessarian]. L'idée est la suivante:

$\Leftarrow$ On utilise un automate à un état qui simule dans sa pile une dérivation top-down de la grammaire en lisant les terminaux au fur et à mesure.

$\Rightarrow$ Plus compliqué: certains non-terminaux de la grammaire encodent un état q et un symbole de pile E. Le rang de ces non-terminaux est $Rang(E) \times card(Q)$ et ses fils seront toutes les variations possibles de la pile sous eux dans tous les états possibles.

Une règle ϵ $q(v, E(x_1...x_m)) \rightarrow q'(v, \pi')$ est traduite de manière assez directe: la tête de lecture (noeud dérivé) passe de q à q', le symbole de E au symbole racine de $\pi'(\pi'_1...\pi'_m)$, et les "paquets" de $card(Q)$ fils représentant $Q \times \pi'_i$ sont changés en des paquets de $card(Q)$ fils représentant Q fois les fils de $\pi'(\pi'_1...\pi'_m)$.

Une règle de lecture $q(f(v_1...v_n), E(x_1...x_m)) \rightarrow f(q_1(v_1, \pi_1), ..., q_n(v_n, \pi_n))$ transforme (q,E) en f, puis choisit l'état q_i comme état de la tête de lecture pour le i^{eme} fils, et le symbole de tête de $\pi_i(\pi'_1...\pi'_m)$ comme symbole.

Mieux encore, ce formalisme de PDTA est équivalent à un autre, encore plus simple, n'utilisant que des symboles de piles d'arité 1, ce qui veut dire que la pile s'apparente à une pile linéaire classique.

Définition. *Un PDTA restreint est un PDTA dont l'alphabet* Π *n'a qu'un symbole d'arité 0,* Z_0*, puis uniquement des symbole d'arité 1.*
Les règles sont de la forme:

- *Règles de lecture:*
  $q(f(v_1...v_n), E(x)) \rightarrow f(q_1(v_1, \pi_1 x), ..., (q_n(v_n, \pi_n x))$
- *Règles* ϵ*:*
  $q(v, E(x)) \rightarrow q'(v, \pi' x)$

Théorème I.2. *Un langage est accepté par un PDTA si et seulement si il est reconnu par un PDTA restreint.*

Démonstration. Voir [Guessarian] pour la démonstration qui démontre que PDTA, PDTA restreint et CFTL rang borné reconnaissent tous trois la même classe de langage.

Le $\Leftarrow$ est évident car tout PDTA restreint est un PDTA.

Le $\Rightarrow$ est plus complexe: on passe par la grammaire context-free équivalente à l'automate qu'on souhaite restreindre. On utilise alors l'état et la pile pour se "rappeler" des transitions top-down que l'on veut simuler dans cette grammaire: en haut de pile, la plus haute, le reste correspond à ce qui restera à faire.

[Guessarian] a prouvé une équivalence entre une grammaire pour décrire et un automate pour reconnaitre, définissant ainsi un formalisme solide pour les CFTL sur les alphabets à rang borné. Cette solution ne se traduit hélas pas directement aux arbres sans rang, il faut donc chercher un formalisme aux propriétés similaires afin de décrire des langages context-free sur les arbres sans rang (par exemple sur les arbres XML).

Part II
CFTL sans rang

3 Arbres sans rang

Bien que disposant de formalismes aux propriétés intéressantes, les arbres de rang borné ne suffisent pas à décrire certains arbres (ex: l'encodage XML d'un fichier HTML est par nature sans rang). On cherche donc à étendre l'approche CFTL aux arbres sans rang.
Les arbres sans rang sont définis dans [TATA] comme un cas général d'arbre étiquetés sur un alphabet sans rang $\mathcal{F}$. On peut les définir ainsi:

Définition. *L'ensemble $\mathcal{T}(\mathcal{F})$ d'arbres sans rang sur $\mathcal{F}$ est le plus petit ensemble tel que:*

- *L'arbre vide $\epsilon \in \mathcal{T}(\mathcal{F})$;*
- *Si $a \in \mathcal{F}$, $a \in \mathcal{T}(\mathcal{F})$;*
- *Si $a \in \mathcal{F}, T_0...T_k \in \mathcal{T}(\mathcal{F})$ non-vides, $a(T_0...T_k) \in \mathcal{T}(\mathcal{F})$.*

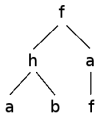
Dans l'arbre $a(T_0...T_k)$, a est la racine.
Les racines de $T_0...T_k$ sont les fils de a: a est leur père.

L'ensemble $\mathcal{H}(\mathcal{F})$ des haies sans rang sur $\mathcal{F}$ est le plus petit ensemble tel que:

- *Si $T \in \mathcal{T}(\mathcal{F})$, $T \in \mathcal{H}(\mathcal{F})$;*
- *Si $T \in \mathcal{T}(\mathcal{F})$, $H \in \mathcal{H}(\mathcal{F})$, $T.H \in \mathcal{H}(\mathcal{F})$.*

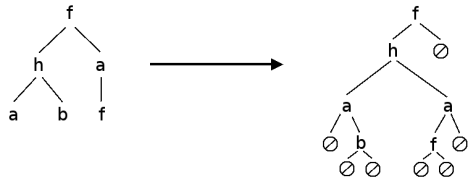
où $\epsilon.H = H.\epsilon = H$

Exemple.



Notons qu'on peut encoder une haie sans rang en un unique arbre binaire: l'encodage First Child Next Sibling (ou FCNS). Le premier fils d'un noeud devient son fils de gauche, son prochain frère (le prochain fils de son père) devient son fils droit.
S'il n'a pas de fils (resp. de frère), son fils gauche (resp. droit) est le symbole $\oslash$ (d'arité 0), ce qui permet un encodage unique et réversible en arbre binaire.

Exemple.



Afin de créer une grammaire Context-Free pour les arbres sans rang, on doit définir une "vue" sur les fils, i.e une manière de désigner la liste des fils qui soit valable quelque soit leur nombre.

On cherche trois propriétés:

- Il doit décrire au moins tous les langages CFTL rang borné, mais aussi tous les langages de $FCNS^{-1}$(CFTL rang borné).
- Les problèmes du vide et de l'appartenance doivent être décidable.
- On doit trouver un formalisme d'automates d'arbres équivalent aux grammaires, afin de reconnaitre les langages décrits.

La première approche (Unary) se base sur une vue unaire des fils.

4 Unary CFTL

Une grammaire unaire est une grammaire d'arbres Context-Free où tous les fils d'un même noeud sont pris comme un tout. Qu'il y ait 0 ou 1 ou k fils, on appliquera les mêmes règles de la même manière.

Définition. *Une grammaire unaire G est la donnée de:*

- *Un alphabet $\mathcal{N}$ de symboles non-terminaux.*
- *Un symbole initial S_0.*
- *Un ensemble de règles Δ.*

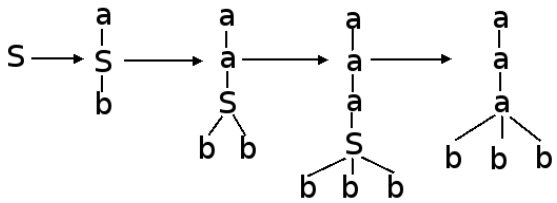
Une règle de Δ est de la forme $N(X) \to H(X)$ où $N \in \mathcal{N}$ et H(X) est une haie sur $\mathcal{H}(\mathcal{F} \cup \mathcal{N} \cup \{X\})$, X ne pouvant être qu'une feuille.

Une dérivation selon cette règle se déroule comme suit: le noeud dérivé N est remplacé par H(X) où X est remplacé à chaque occurrence par la haie des fils de ce N.
Une haie de $\mathcal{H}(\mathcal{F})$ est reconnue par G si on l'obtient après une ou plusieurs dérivations selon les règles de Δ depuis S_0.
Le langage L(G) est l'ensemble des haies de $\mathcal{H}(\mathcal{F})$ reconnues par G.

Exemple. .

- $\mathcal{F} = \{a, b\}$
- $\mathcal{N} = \{S\}$, *non-terminal initial.*
- Δ :
  $S(X) \to a(S(X.b))$
  $S(X) \to X$

Exemple de dérivation:

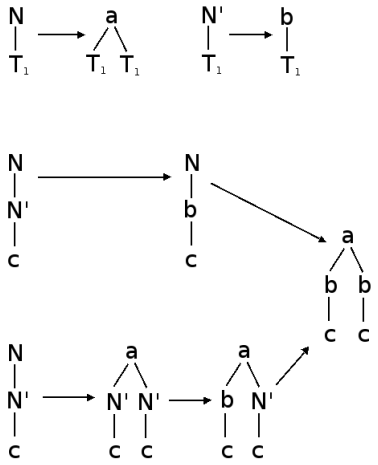


On démontrera que sur ces grammaires, on peut dériver en premier les noeuds les plus hauts (dérivation Top-Down) sans changer le langage reconnu.

Lemme II.1. *On peut décider de ne dériver que les non-terminaux les plus hauts, i.e des non-terminaux qui n'ont aucun ancêtre non-terminaux, sans modifier le langage reconnu par une grammaire.*

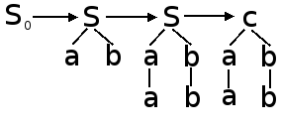
Démonstration. Cette contrainte ne crée aucune nouvelle dérivation, aussi le langage obtenu est un sous-ensemble du langage original, ou égal au langage original.
Afin de rendre une dérivation top-down, on choisit un non-terminal le plus haut N. Sa dérivation $N(X) \to H(X)$ est précédée de la dérivation de certains de ses descendants, c'est-à-dire:
$N(H') \to N(H'') \to H(H'')$, avec
$H' \to H''$ la transformation des descendants de N.
On peut pour rétablir l'ordre top-down effectuer $N(X) \to H(X)$ en premier, puis appliquer les dérivations menant de H' à H" à chaque occurrence de H' créée par la dérivation de N.
On répète ce processus jusqu'à ce que la dérivation soit top-down.
Afin de se convaincre de la terminaison de ce processus, on peut considérer le multi-ensemble des dérivations de la suite initiale de dérivations, ordonnée par priorité top-down, à savoir, si une dérivation doit être effectuée avant une autre (si elle crée le non-terminal dérivé par cette autre), elle est plus "grande". Si une dérivation a lieu sous une autre, elle est plus petite.
Chaque étape du processus multiplie, au pire, par un nombre fini (nombre de X dans la règle de Δ) le nombre des dérivations plus petites, et peut se débarrasser de la dérivation triée. On fait donc décroitre le multi-ensemble, le processus se termine donc.

Exemple.



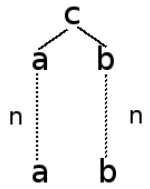
Le problème du formalisme Unary est qu'il ne permet pas de décrire tous les langages de CFTL rang borné:
Soit la grammaire rang borné G_0:

- $N = S_0, S$
- Δ :
  $S_0 \to S(a, b)$
  $S(x_1.x_2) \to S(a(x_1).b(x_2))$
  $S(x_1.x_2) \to c(x_1.x_2)$

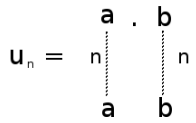


Théorème II.1. *Aucune grammaire unaire ne reconnait* $L(G_0)$

Note: $L(G_0)$ est l'ensemble:



Démonstration. On démontrera qu'aucune grammaire unaire ne peut reconnaitre L' l'ensemble des



Ces deux résultats sont équivalents comme démontré dans le lemme suivant:

Lemme II.1.1. *Il existe une grammaire unaire reconnaissant $L(G_0)$ $\Leftrightarrow$ Il existe une grammaire unaire reconnaissant L'.*

Démonstration.

- $\Rightarrow$ Si $G = \{\mathcal{N}, S_0, \Delta\}$ reconnait $L(G_0)$ sur $\{a, b, c\}$, alors $G' = \{\mathcal{N} \cup \{c\}, S_0, \Delta \cup \{c(X) \to X\}\}$ reconnait L' sur $\{a, b\}$.
- $\Leftarrow$ Si $G = \{\mathcal{N}, S_0, \Delta\}$ reconnait L', alors $G' = \{\mathcal{N} \cup \{S_0'\}, S_0', \Delta \cup \{S_0'(X) \to c(X)\}\}$ reconnait $L(G_0)$.

Afin de prouver le théorème II.1, on supposera que l'on connait une grammaire G qui reconnait L'. Étudions le chemin de dérivation $u_n = a^n.b^n$.
On considèrera une dérivation top-down. Arrivera un moment où a^n et b^n vont se séparer, c'est-à-dire que juste avant cela, l'arbre donnant u_n sera N(H) où on aura pas deux sous-arbres de N(H) donnant l'un a^n et l'autre b^n. Juste après cela, il y aura deux arbres $T_a(H)$ et $T_b(H)$ tels que $T_a(H)$ donne a^n et $T_b(H)$ donne b^n.
Cette séparation existe pour $n > 1$ car ni $S_0(\epsilon)$ ni ϵ ne donnent a^n ou b^n, alors que pour u^n en fin de dérivation, a^n et b^n se "donnent" eux mêmes. Le passage de la première à la deuxième situation est la dérivation de séparation.
T_a et T_b sont les sous-arbres de la règle de Δ utilisée lors de la séparation tels que $T_a(H)$ donne a^n et $T_b(H)$ donne b^n.

Vu que la dérivation est top-down, on peut dériver T_a et T_b avant de dériver H.
$T_a(H) \to T_a'(H)$ et $T_b(H) \to T_b'(H)$
$T_a'(H)$ peut être:

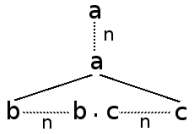
- a^n lui-même
- $a^{n-i}(H)$ où H donne a^i

Le cas $T_b'(H)$ est symétrique, mais H ne peut se dériver à la fois en a^i et b^i, donc l'un des deux doit se dériver directement en a^n ou b^n, ce qui veux dire qu'on ne peut les utiliser comme T_a et T_b que pour ce u_n particulier.

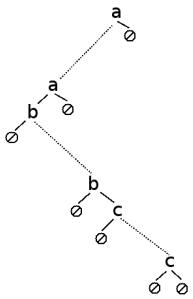
Δ est finie, donc les sous-arbres T_a et T_b le sont aussi, mais on doit en fournir une infinité, pour l'infinité de u_n à dériver, aussi il faudrait utiliser un T_a et T_b une infinité de fois, d'où une contradiction.

Ainsi aucune grammaire ne peut reconnaitre L', ni donc reconnaitre $L(G_0)$.

L'approche Unary manque donc d'expressivité pour satisfaire nos critères. Cependant, elle suffit pour créer un langage dont l'encodage FCNS n'est pas CFTL rang borné.
$\mathcal{N} = \{S\}$, S est le non-terminal initial.
Δ :
$S(X) \to a(S(b.X.c))$
$S(X) \to X$



L'encodage FCNS de ce langage est:



Si ce langage est CFTL rang borné on peut trouver un automate d'arbre à pile qui le reconnait.

Cet automate peut être modifié en un automate à pile de mots reconnaissant $a^n c^n b^n$, langage non-CF, ce qui n'est pas possible.

De plus, les problèmes du vide et de l'appartenance sont décidables:

Théorème II.2. *Soit une grammaire unaire $G = \{\mathcal{N}, S_0, \Delta\}$, on peut décider si L(G) est vide ou non.*

Démonstration. L'algorithme proposé est en quelque sorte "glouton", il dérive tant qu'il peut sans créer un seul nouveau non-terminal. Quand il ne peut plus, il a démontré l'existence d'un "piège", c'est-à-dire que chaque dérivation a créé un nouveau non-terminal. Si S_0 fait partie de ce piège, L(G) est vide; sinon, L(G) est non-vide, et l'algorithme fournit un exemple.

On va transformer G à de nombreuses reprises sans changer son problème du vide.

Première étape

Il s'agit de remplacer certains non-terminaux N en H tel que:
$N(X) \to H \in \Delta$, où H ne contient ni non-terminaux, ni X.
On remplace chaque N par son image H dans le membre droit de chaque règle de Δ. Si $N = S_0$, alors $H \in L(G)$.
Sinon, on supprime définitivement N de la grammaire, et les règles $N(X) \to H(X)$ de Δ.

- Cette nouvelle grammaire force une certaine dérivation de N dans G: sont langage est inclus dans L(G). Si L(G) est vide, ce langage aussi.
- Si L(G) n'est pas vide, on considère une dérivation top-down de S_0 à un mot de L(G). A chaque fois qu'on rencontre $N(H')$ on peut tout aussi bien le dériver en H, ce qui ne gênerait en rien la terminaison de la dérivation car H ne contient ni non-terminaux ni X. La dérivation obtenue peut être transformée en une dérivation dans la nouvelle grammaire, dont le langage n'est donc pas vide.

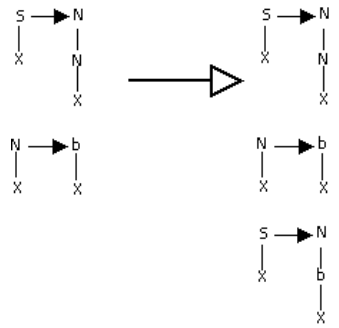
Cette nouvelle grammaire devient le nouveau G et l'on répète cette première étape jusqu'à ce qu'il n'y ait plus de N et H tels que:
$N(X) \to H \in \Delta$, où H ne contient ni non-terminaux ni X.

Note: cette boucle se termine, car elle détruit à chaque fois un non-terminal, et qu'il n'y en a qu'un nombre fini.

Deuxième étape

La deuxième étape consiste à "tamiser" la nouvelle grammaire afin d'obtenir de nouvelles occasion d'utiliser une dérivation qui ne crée aucun non-terminal. Dans ce but on choisit les règles $N(X) \to H(X)$ où H ne contient pas de non-terminaux mais une ou plusieurs occurrences de X.
Si S_0 en a une, alors $H(\epsilon) \in L(G)$.
Sinon on choisit une telle règle au plus par non-terminal N, et on l'appliquera dans les règles de Δ où l'un de ces N est un terminal le plus bas, i.e où il apparait comme $N(H')$, H' ne contenant aucun non-terminaux.

Exemple:



Cependant, on ne supprime ni les non-terminaux, ni les règles originales: les règles créées sont ajoutées à Δ.
Cette nouvelle grammaire est équivalente à la précédente, car les règles créées ne font qu'exprimer la possibilité d'utiliser une des dérivations choisies. Leurs problèmes du vide est donc le même.
La nouvelle grammaire devient G et l'on répète cette seconde étape, en gardant les règles choisies précédemment, ajoutant possiblement de nouveaux non-terminaux N à la sélection si l'on a créé une règle $N(X) \to H(X)$ où H ne contient aucun non-terminal mais une occurrence de X ou plus.
On n'a besoin de vérifier que les règles récemment créées pour l'ancienne sélection, alors que les nouveaux venus devront parcourir toutes les règles de G.

Note: ce processus s'achève: chaque règle de Δ ne peut être modifiée qu'une fois par non-terminal qu'elle contient, car une modification supprime un non-terminal mais n'en crée pas de nouveau.

On répète cette alternance des étapes un et deux jusqu'à la stabilisation.
Ce processus se termine car à chaque première étape utile, on supprime un non-terminal. Il n'y en a donc qu'un nombre fini. Considérons donc la dernière première étape utile. Après cela, la seconde étape va se stabiliser, et le processus sera fini.

Lemme II.2.1. *A la fin de ce processus, si S_0 a une dérivation $S_0(X) \to H(X)$ où H ne contient aucun non-terminal alors $H(\epsilon) \in L(G)$.*
Sinon, S_0 fait partie d'un piège, un ensemble de non-terminaux $N \in \mathcal{N}_{Trap}$ qui n'ont aucune dérivation $N(X) \to H(X)$ où H ne contient aucun non-terminal et tels que chaque dérivation partant d'un $N \in \mathcal{N}_{Trap}$ contient un non-terminal de $\mathcal{N}_{Trap}$.

Démonstration. Le premier point est direct: on applique $S_0(X) \to H(X)$ à $S_0(\epsilon)$
Le second point vient de la stabilisation du processus. On a $\mathcal{N}_{Trap}$ l'ensemble

de N n'ayant aucune dérivation $S_0(X) \to H(X)$ où H ne contient aucun non-terminal. S_0 en fait partie.
Soit $N(X) \to H(X)$ où $N \in \mathcal{N}_{Trap}$. H doit au moins contenir un non-terminal, ou N n'est pas dans $\mathcal{N}_{Trap}$. Soit un non-terminal le plus bas N_0. S'il avait une règle $N_0(X) \to H(X)$ où H ne contient aucun non-terminal, alors le processus l'aurait éliminé; donc N_0 n'en a pas et est dans $\mathcal{N}_{Trap}$.

Si $H(\epsilon) \in L(G)$ alors le langage de cette grammaire (et celui de la grammaire d'origine) n'est pas vide. Ce mot appartient même au langage d'origine.
Sinon, toute dérivation crée au moins un $N \in \mathcal{N}_{Trap}$ et ne se termine donc jamais.
En effet, comme montré précédemment, toute dérivation partant de S_0 contient un non-terminal de $\mathcal{N}_{Trap}$ et tout non-terminal n'appartenant pas à $\mathcal{N}_{Trap}$ n'est pas le plus bas. On montre que cette propriété se préserve au fil d'une dérivation top-down, par récurrence sur les dérivations simples.

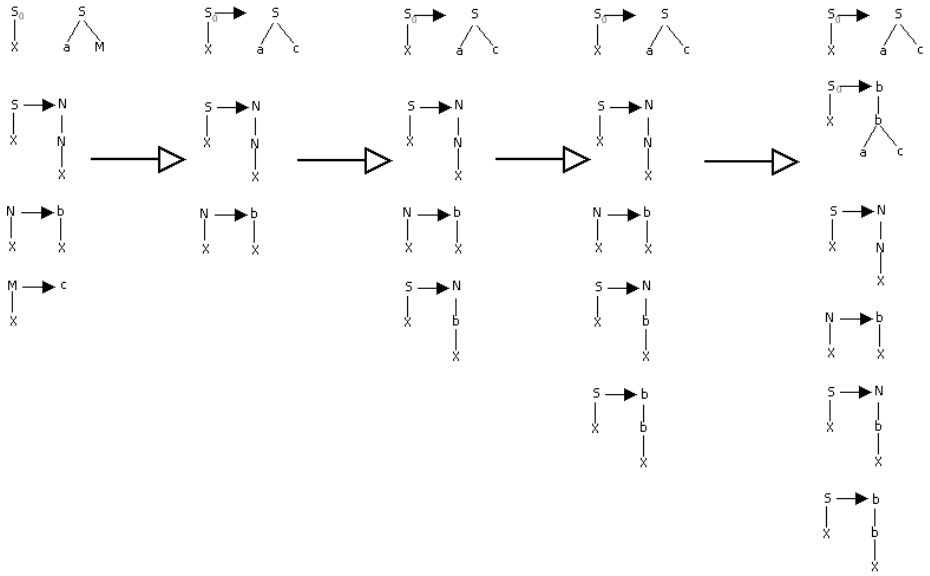
- Si le non-terminal dérivé $N \in \mathcal{N}_{Trap}$, alors la dérivation qui s'en suit crée au moins un non-terminal de $\mathcal{N}_{Trap}$ et tout non-terminal n'appartenant pas à $\mathcal{N}_{Trap}$ n'est pas le plus bas, qu'il soit créé ou recopié (par hypothèse);

- Si le non-terminal N n'est PAS dans $\mathcal{N}_{Trap}$, alors ce n'est pas un non-terminal le plus bas (par hypothèse), et ses fils ont un non-terminal de $\mathcal{N}_{Trap}$. Une dérivation de ce noeud est $N(X) \to H(X)$ où H ne contient aucun non-terminal mais au moins un X ou qui contient au moins un non-terminal, et dont les non-terminaux les plus bas sont des $N_0 \in \mathcal{N}_{Trap}$ en suivant le même raisonnement que dans le lemme plus haut: le contraire impliquerait un processus incomplet.
  Si la dérivation suit $N(X) \to H(X)$ où H ne contient aucun non-terminal mais au moins un X, ce X représente les fils de N, qui respectent la propriété requise.
  Si la dérivation suit $N(X) \to H(X)$ où H contient au moins un non-terminal, et dont les non-terminaux les plus bas sont des $N_0 \in \mathcal{N}_{Trap}$, la propriété est directement conservée.

Ainsi la propriété est conservée tout au long de la dérivation ce qui implique qu'un $N \in \mathcal{N}_{Trap}$ sera toujours présent, et donc que la dérivation ne peut terminer.

Cet algorithme permet donc de décider le problème du vide des grammaires unaires.

Exemple.



Nous devons cependant envisager un formalisme plus expressif.

Part III
Head:Tail CFTL

5 Formalismes non-discriminant et discriminant

Une vue unaire des fils est trop peu expressive pour décrire CFTL rang fini. Afin de corriger cela, on introduit un autre type de grammaire basée sur une vue plus fine: la vue Head:Tail.

L'idée est que l'on peut désormais avoir accès individuellement au premier fils d'un noeud (Head), puis au reste de fils tous ensemble (Tail).
On peut avoir deux approches du cas où un noeud n'a aucun fils, ce qui distingue deux formalismes:

- Une première approche consiste à considérer que tout noeud a une infinité de fils valant ϵ à la fin de sa liste de fils non-vides: c'est une approche non-discriminante Head:Tail.
- Une seconde approche consiste à différentier les noeuds avec ou sans fils, avec deux jeux de règles mutuellement exclusifs: c'est une approche discriminante Head:Tail.

On montrera que la seconde approche est strictement plus puissante que la première, mais aussi que ces deux formalismes décrivent CFTL rang borné, $FCNS^{-1}$(CFTL rang borné) et sont même strictement plus expressifs que ces derniers.

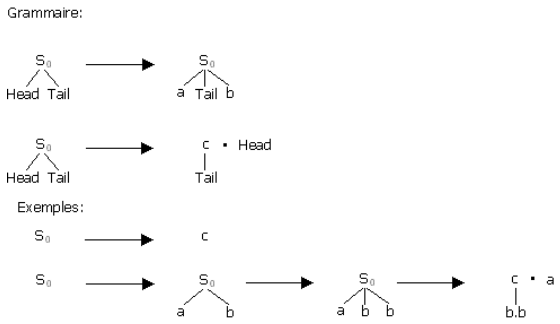
Définition. *Une grammaire non-discriminante Head:Tail G est définie sur un alphabet $\mathcal{F}$. C'est la donnée de:*

- *Un alphabet de non-terminaux $\mathcal{N}$;*
- *Un non-terminal initial S_0;*
- *Un ensemble de règles Δ;*

Une règle de Δ est de la forme:
$N(Head : Tail) \to H$
où $N \in \mathcal{N}$ et H est une haie sur $\mathcal{F} \cup \mathcal{N} \cup \{Head, Tail\}$, tel que Head et Tail ne peuvent être que des feuilles de H.

La dérivation d'un noeud selon cette règle consiste à remplacer N par H dans lequel Head est remplacé par le premier fils de N (ou ϵ s'il n'y en a pas) et Tail est remplacé par le reste des fils de N (ou ϵ s'il n'y en a pas).
Un chemin de dérivation sur G est une succession de dérivations, partant de $S_0(\epsilon)$ et arrivant à une haie de $\mathcal{H}(\mathcal{F})$.
Une haie est reconnue par G si c'est l'arrivée d'un chemin de dérivation. Le langage $L(G)$ reconnu par G est l'ensemble des haies reconnues par G.

Exemple.



La différence entre le formalisme discriminant ou non-discriminant est l'existence pour le second de deux jeu de règles, l'un pour les noeuds avec fils, l'autre pour les noeuds sans fils.

Définition. *Une grammaire Head:Tail discriminante G est définie sur un alphabet $\mathcal{F}$.*
C'est la donnée de:

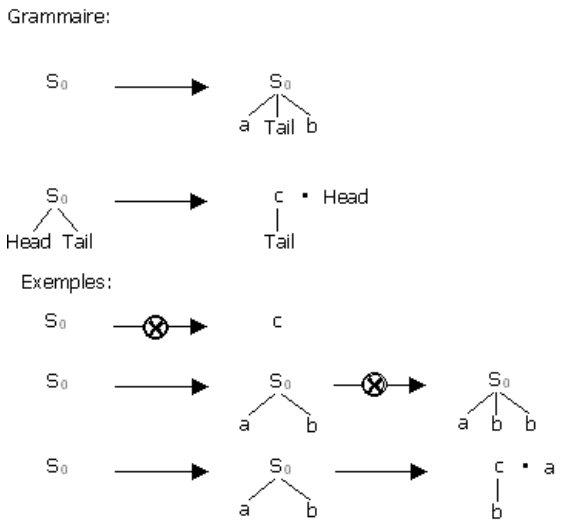
- *Un alphabet de non-terminaux $\mathcal{N}$;*
- *Un non-terminal initial S_0;*
- *Un ensemble de règles Δ pour noeuds avec fils;*
- *Un ensemble de règles Δ_ϵ pour noeuds sans fils;*

Les règles de Δ sont de la forme:
$N(Head : Tail) \to H$
où $N \in \mathcal{N}$ et H est une haie sur $\mathcal{F} \cup \mathcal{N} \cup \{Head, Tail\}$, telle que Head et Tail ne peuvent être que des feuilles.

Les règles de Δ_ϵ sont de la forme:
$N(\epsilon) \to H$
où $N \in \mathcal{N}$ et H est une haie sur $\mathcal{F} \cup \mathcal{N}$.

La dérivation d'un noeud suivant une règle de Δ consiste à remplacer N par H, où Head devient le premier fils du noeud N et Tail devient le reste de ses fils (ou ϵ s'il n'y en a pas). Pour utiliser une telle règle, N doit avoir au moins un fils.
La dérivation d'un noeud suivant une règle de Δ_ϵ consiste à remplacer N par H. Pour utiliser une telle règle, N ne doit avoir aucun fils.
Un chemin de dérivation dans G est une succession de dérivations, partant de $S_0(\epsilon)$ et terminant sur une haie sans non-terminaux.
Une haie est reconnue par G si un chemin de dérivation s'y termine.
Le langage reconnu par G est l'ensemble des haies reconnues par G.

Exemple.



Lemme III.1. *Le formalisme discriminant est au moins plus puissant que le formalisme non-discriminant.*

Démonstration. Soit une grammaire non-discriminante $G = (\mathcal{N}, S_0, \Delta)$
Une grammaire discriminante équivalente serait: $G' = (\mathcal{N}, S_0, \Delta, \Delta_\epsilon)$ où Δ_ϵ est l'ensemble des règles de Δ où l'on remplace chaque occurrence de Head et Tail par ϵ, i.e:
Si $N(Head : Tail) \to H \in \Delta$ alors
$N(\epsilon) \to H[Head \leftarrow \epsilon, Tail \leftarrow \epsilon] \in \Delta_\epsilon$

Une dérivation dans G peut être remplacé par une dérivation dans G' en la remplaçant par son clone, dans Δ si le noeud dérivé a un ou des fils, dans Δ_ϵ sinon.

Une dérivation dans G' peut être remplacé par une dérivation dans G en la remplaçant par la règle de Δ dont elle dérive.

Ainsi, s'il existe un chemin de $S_0(\epsilon)$ à une haie H dans une de ces grammaires, il en existe un dans l'autre; ces grammaires sont équivalentes.

Le formalisme discriminant est donc capable de simuler le non-discriminant, ce qui le rend plus puissant.

Aussi quand il s'agira de prouver que ces formalismes sont plus puissants que CFTL rang borné et que l'encodage FCNS de leurs langages est strictement plus riche que CFTL rang borné, on se contentera de le prouver pour le formalisme non-discriminant.

6 Propriétés

Théorème III.1. *Soit $G = (\mathcal{N}, S_0, \Delta)$ une grammaire CFTL rang borné. Il existe G' une grammaire Head:Tail non-discriminante qui reconnait le même langage.*

Démonstration. Partant des transitions $N(x_1, ..., x_k) \to T(x_1, ..., x_k) \in \Delta$ l'idée est de remplacer chaque occurrence de x_n dans chaque T de chaque règle de Δ par:
$X_n(Head : Tail)$, où $X_1(Head : Tail) \to Head$ et
$X_{n+1}(Head : Tail) \to X_n(Tail)$
$X_n(Head : Tail)$ se comporte exactement de la même manière que x_n, car c'est toujours un arbre qui au final se dérivera en une dérivation du n^{eme} fils de la liste Head:Tail.

On peut le prouver récursivement depuis X_1 pour tout X_n:

- $X_1(H)$ est toujours un arbre, et se dérive en une dérivation du premier fils de H:
  $X_1(H)$ est, tel quel,un arbre. Pendant que H se dérive en H', il garde le même "profil" horizontalement, vu que cette dérivation se fait selon des règles CFTL rang borné. Le premier fils de H' à l'issue des dérivations est le premier fils de H qui a subi toutes les dérivations qui le concernent dans $H \to H'$.
  Vu que $X_1(H')$ est remplacé par le premier fils de H', c'est toujours un arbre, qui de plus, est une dérivation du premier fils de H.

- Si $X_n(H)$ est toujours un arbre, et se dérive en une dérivation du n^{eme} fils de H, alors $X_{n+1}(H)$ est toujours un arbre, et se dérive en une dérivation du $n + 1^{eme}$ fils de H:
  $X_{n+1}(H)$ est, tel quel,un arbre. On utilise la même méthode que plus haut pour affirmer qu'ultimement il se dérivera en $X_n(H')$ où H' est une dérivation de H sans son premier fils.
  On utilise l'hypothèse, $X_n(H')$ se dérive en une dérivation du n^{eme} fils dans H', i.e une dérivation de dérivation du $n + 1^{eme}$ fils dans H, i.e une dérivation du $n + 1^{eme}$ fils dans H.
  Ainsi $X_{n+1}(H)$ est toujours un arbre, et se dérive en une dérivation du $n + 1^{eme}$ fils dans H.

On peut donc construire G' ainsi:
$G' = \{\mathcal{N} \cup \{X_i, i \leq n\}, S_0, \Delta' \cup \Delta_X\}$, où:

- n est l'arité maximum de $\mathcal{F}$;

- Δ_X est l'ensemble des règles:
  $X_{n+1}(Head : Tail) \to X_n(Tail)$ et
  $X_1(Head : Tail) \to Head$

- Δ' est l'ensemble des règles de Δ où chaque x_i est remplacé par $X_i(Head : Tail)$ dans les membres droits.

Cette grammaire est par construction équivalente à G.

On conclut donc que les formalismes Head:Tail sont capable de simuler le formalisme CFTL rang borné.

On prouve ensuite que $FCNS^{-1}$(CFTL rang borné) est strictement moins puissant que ces formalismes.

Théorème III.2. *Soit L un langage telle que la grammaire de rang borné $G = \{N, S_0, \Delta\}$ reconnaisse FCNS(L). Alors il existe G' une grammaire non-discriminante qui reconnait L.*

Démonstration. Si L est reconnu par une grammaire de rang borné, il l'est également par un PDTA restreint $\mathcal{A} = \{Q, \mathcal{F}, \Pi, q_0, Z_0, R\}$.
L'idée de la démonstration est de simuler cet automate à l'aide d'une grammaire $G = \{[Q \times (\mathcal{F} \cup \{\oslash\})] \cup \Pi \cup \{B, S_0\}, S_0, \Delta\}$ sur l'alphabet $\mathcal{F}$
qui "défait" l'encodage FCNS au cours de la dérivation.
La simulation dérivera la tête de lecture de l'automate, "inventant" l'arbre au fur et à mesure, tout en stockant la pile sous la tête de lecture.

On introduit tout d'abord deux non-terminaux spéciaux:

- B est un bloc "toxique": Il ne peut pas se dériver de lui-même est doit, pour disparaître, être supprimé par un noeud supérieur.
- E est un noeud auto-effaçant:
  $E(Head : Tail) \to Head.Tail$

Ces noeuds serviront dans la simulation des tests décrits plus bas.

La tête de lecture
Elle débute en un $(q_0, a)(Z_0), a \in (\mathcal{F} \cup \{\oslash\})$, en inventant la racine de l'arbre et en commençant en l'état q_0 avec une pile Z_0. $Q_f \times \oslash$ sont les états terminaux, et se dérivent en ϵ. Le reste du temps la tête de lecture est dans un "état" de $Q \times \mathcal{F}$
Afin de simuler une transition, la tête de lecture doit lire le premier membre de sa pile (ce qui correspond à un test décrit plus bas), puis soit changer son état et sa pile de manière statique ou écrire la lettre en cours de lecture et déplacer la tête de lecture à la fois vers le bas (pour ses fils, la partie gauche de l'encodage FCNS) et à sa droite (pour ses frères, la partie droite de l'encodage FCNS).

Exemple. *La transition $q(v, Z_i(x_1)) \to q'(v, \pi(x_1))$*
se traduit en $(q, a)(Head : Tail) \to (q', a)(\pi.Tail).Test_{i+2}(Head)$ pour chaque $a \in (\mathcal{F} \cup \{\oslash\})$.
La transition $q(a(v_1, v_2), Z_i(x)) \to a(q_1(v_1, \pi_1(x)), q_2(v_2, \pi_2(x)))$
se traduit en $(q, a)(Head : Tail) \to [a((q_1, b)(\pi_1.Tail))].(q_2, c)(\pi_2.Tail).Test_{i+2}(Head)$ pour chaque $b, c \in (\mathcal{F} \cup \{\oslash\})$

La pile

La pile est stockée à l'horizontale sous la tête de lecture. Chaque $Z_i \in \Pi$ possède une règle qui lui est propre qui lui permet de se dériver en B^{i+2}, ce qui servira pour la section **Test** de cette simulation.
La tête de la pile est le premier fils de la tête de lecture, accessible via Head. Ajouter une pile π consiste à l'ajouter à gauche de la pile courante. Le pop consiste à ne pas replacer Head à la gauche de Tail.

Tests

La tête de lecture n'ayant pas de véritable pouvoir de lecture, on établit des tests afin de vérifier à chaque étape que la transition simulée est utilisée de manière légitime.

$Test_n(Head : Tail)$: ce test n'est passé avec succès que par les haies qui peuvent se dériver en exactement n arbres. On utilisera des non-terminaux spéciaux $P_{i,j}(Head : Tail)$.
$Test_n(Head : Tail) \rightarrow P_{n+1,n+1}(Head.Tail.E.B^{n+1})$

Première étape:
La première étape consiste à choisir le $n+1^{eme}$ fils et tester qu'il peut se supprimer.
Pour n>1, $P_{n,m}(Head : Tail) \rightarrow P_{n-1,m}(Tail)$
$P_{1,m}(Head : Tail) \rightarrow Head.P_{0,m}(Tail)$
Nb: ici, il suffit de vérifier que Head peut se dériver complètement; en règle générale, on peut vouloir tester qu'il se dérive bien en ϵ, par exemple en gardant le Tail de Head.B

Deuxième étape:
La deuxième étape consiste à vérifier qu'il n'y a plus de fils après cela et qu'il ne reste que les B^{n+1} que l'on a nous-même ajoutés.
Pour m>1 $P_{0,m}(Head : Tail) \rightarrow P_{0,m-1}(Tail)$
$P_{0,1} \rightarrow Tail$

Si une haie peut se dériver en exactement n arbres, la première étape supprimera ces n fils et choisira E comme $n+1^{eme}$ fils. La deuxième étape supprimera les B^{n+1} que l'on a ajoutés.

Si une haie se dérive en k<n arbres, la première étape supprimera les k arbres puis d'autres fils jusqu'au n^{eme}, pour choisir un B dans B^{n+1}. Le test échoue.

Si une haie ne peut pas se dériver en exactement n arbres, mais en k>n arbres, la première étape supprimera n fils et choisira le $n+1^{eme}$ fils. S'il ne peut se dériver en ϵ, le test échoue. En supposant qu'il peut, alors il reste un fils qui ne le peut pas, sinon la haie pourrait se dériver en exactement n arbres. La seconde étape ne pourra alors pas supprimer tous les B^{n+1}

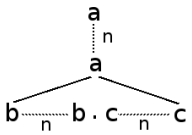
en fin de liste, car il doit également s'occuper des fils en excès. Le test échoue.

Pour vérifier que le symbole de tête de la pile actuelle est bien Z_i, il suffit de faire $Test_{i+2}(Head)$: Z_i est le seul symbole à pouvoir réussir ce test.

Vu que l'on "force" une dérivation "Top-Down", les tests ne peuvent être supprimés en cas d'échec, assurant ainsi qu'une dérivation réussie ne simule que des mouvements justifiés, et donc que la haie créée est le déroulé d'un arbre de FCNS(L). Aussi G' reconnait $FCNS^{-1}(L)$.

Notons que cette preuve montre un moyen de simuler un PDTA restreint, ce qui constitue un autre moyen de simuler CFTL rang borné grâce à l'équivalence prouvée dans [Guessarian].

Afin de prouver que $FCNS^{-1}$(CFTL rang borné) est strictement moins puissant, on peut utiliser le même exemple que pour Unranked:



Cependant, les problèmes du vide et de l'appartenance sont indécidables pour les formalismes discriminant et non-discriminant.

On le prouve en montrant comment réduire le problème d'accès à un état dans une machine de Minski à deux compteurs à ces problèmes.

Théorème III.3. *Soit M une machine de Minski à deux compteurs, q un état de M. On peut calculer une grammaire G Head:Tail non-discriminante telle que L(G) non-vide $\Leftrightarrow$ q est accessible dans M.*

Le problème d'accessibilité étant lui-même indécidable, cette réduction prouve l'indécidabilité du problème du vide pour les formalismes Head:Tail.

Démonstration. Il s'agit de simuler une machine de Minski, tout en créant des test afin de s'assurer de la validité du chemin simulé. Si un test échoue, il met en échec la dérivation dans son ensemble. Si (et seulement si) l'état q est accessible dans M, il existe une dérivation dans G, donc L(G) non-vide.

On introduit les non-terminaux suivants:

- B est un bloc "toxique": Il ne peut pas se dériver de lui-même est doit, pour disparaître, être supprimé par un noeud supérieur.
- E est un noeud auto-effaçant:
  $E(Head : Tail) \rightarrow Head.Tail$
- Token est un marqueur utilisé dans certains tests plus loin:
  $Token(Head : Tail) \rightarrow B.B$

Première étape: Simuler la tête de lecture

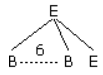
La tête de lecture est modélisée par son état (un non-terminal de la grammaire). Ses deux fils sont les deux compteurs. C'est le premier noeud créé par S_0, et elle débute dans l'état initial de la machine de Minski, avec les deux compteurs à zéro.
Simuler une transition revient à dériver la tête de l'état de départ vers l'état d'arrivée, et à appliquer à chaque fils/compteur l'action qui correspond. La tête de lecture génèrera également les tests nécessaires à la validité de la dérivation, au niveau de ses frères.

Exemple. *La transition* $q \overset{x=0}{\rightarrow} q', x++, y--$ *devient:*
$q(Head : Tail) \rightarrow q'(Inc(Head).Dec(Tail)).TestZero(Head).NonZero(Tail).Test_2(Head.Tail)$
qu'on peut interpréter comme:
$q(Head : Tail) \rightarrow q'((x++).(y--)).TestZero(x).NonZero(y).Test_2(x.y)$
Note: TestZero, NonZero et $Test_2$ *sont décrits dans la partie Test.*

Deuxième étape: Simuler les compteurs

Afin de pouvoir leur passer correctement leurs actions, on impose des compteurs qu'ils soient constamment des arbres non-vides quand ils se trouvent sous la tête de lecture. Un compteur à zéro ne peut pas être ϵ.

On choisira $E(B^6E)$:



comme compteur à zéro pour des raisons décrites plus loin.

L'incrémentation d'un compteur est interprétée par l'empilement d'un non-terminal Inc qui se dérive en triplant le compteur sous lui:
$Inc(Head : Tail) \rightarrow Head.Tail.Head.Tail.Head.Tail$

La décrémentation d'un compteur est interprétée par l'empilement d'un non-terminal Dec qui se dérive en quintuplant le compteur sous lui:
$Dec(Head : Tail) \rightarrow Head.Tail.Head.Tail.Head.Tail.Head.Tail.Head.Tail$

Le non-terminal E ne représente aucune action:
$E(Head : Tail) \rightarrow Head.Tail$

Le compteur vaut zéro quand une fois complètement dérivé il représente le compteur de base répliqué à une puissance de quinze.

Exemple. $Dec(Dec(Inc(Inc(E(B^6E))))) \rightarrow$
$Dec(Inc(Inc(E(B^6E))))^5 \rightarrow$
$Inc(Inc(E(B^6E)))^{25} \rightarrow$
$Inc(E(B^6E))^{75} \rightarrow$
$E(B^6E)^{225} = E(B^6E)^{15^2}$

Troisième étape: Contrôler avec des tests

On n'a besoin que de quelques tests pour certifier le bon fonctionnement de la dérivation:

- A chaque test à zéro lors d'une transition, on doit bien entendu vérifier que le compteur concerné vaut bien zéro (TestZero).
- A chaque décrémentation de compteur, on dot vérifier que le compteur concerné ne vaut PAS zéro (NonZero).
- Les compteurs peuvent se dériver en haies de plus d'un arbre et empêcher la distribution correcte des actions: afin de prévenir cela on testera à chaque étape qu'il y a exactement deux fils sous la tête de lecture ($Test_2$).

Un test est valide si et seulement s'il se dérive en ϵ.
Une fois que l'on aura implémenté correctement ces trois tests, il suffira de les produire au bon moment.

- $Test_n(Head : Tail)$: Ce test n'est passé que par les haies qui peuvent se dériver en exactement n arbres. On peut utiliser les tests décrits dans la démonstration du Théorème III.2 .

On utilise $Test_2$ à chaque transition de la machine de Minski pour vérifier que les compteurs sous la tête de lecture sont toujours des arbres.

- Le test à zéro est plus délicat: il faut s'assurer qu'on dérive entièrement le compteur, puis vérifier que sa valeur est bien zéro.

  – La première étape consiste à vérifier que le compteur est entièrement développé. On n'acceptera que les compteurs sous la forme $(B^6E)^n$.
  $TestZero(Head : Tail) \rightarrow Div_7(Head.Tail.Token)$
  Le rôle de Div7 est de vérifier qu'on a bien un compteur de forme $(B^6E)^n$
  $Div_7(Head : Tail) \rightarrow Div_{7[6]}(Tail)$
  $Div_{7[6]}(Head : Tail) \rightarrow Div_{7[5]}(Tail)$
  ...
  $Div_{7[1]}(Head : Tail) \rightarrow (Div_7(Tail.B)).Head$
  Avec 7 dérivations successives on obtient:
  $Div_7(Head_1...Head_7 : Tail) \rightarrow Div_7(Tail.B).Head_7$

  Si $Head_7$ peut se supprimer de lui-même, c'est $E(\epsilon)$. Ainsi de $Head_1$ à 6 on doit avoir des B car $E(\epsilon)$ n'apparait que dans B^6E.

  On achève la boucle en récupérant Token, le marqueur:
  $Div_7(Head : Tail) \rightarrow TestZero_2(Tail).Test_2(Head)$

  $Test_2$(Head) confirme qu'on a récupéré Token, qui est le seul à se dériver en 2 arbres. Le processus s'achève et on est sûr d'avoir un compteur de forme $(B^6E)^n$
  Notons qu'on l'a réduit en B^n au cours du processus.

  – La seconde étape vérifie que la valeur du compteur est bien 0, ce qui revient à vérifier que n est une puissance de 15.
  $TestZero_2(Head : Tail) \rightarrow Div_{15}(Head.Tail.Token)$

  Le mécanisme de la division est le même, mais l'on a plus besoin de tester la nature des fils: ils sont tous B.
  $Div_{15}(Head : Tail) \rightarrow Div_{15[14]}(Tail)$
  $Div_{15[14]}(Head : Tail) \rightarrow Div_{15[13]}(Tail)$
  ...
  $Div_{15[1]}(Head : Tail) \rightarrow Div_{15}(Tail.B)$

  Une fois la division par 15 achevée, on récupère Token, et on recommence.
  $Div_{15}(Head : Tail) \rightarrow TestZero_2(Tail).Test_2(Head)$

On réitère cette opération jusqu'à ce qu'il ne reste qu'un seul B (ce qui n'arrive que si la valeur du compteur est 0).
Quand il ne reste plus qu'un B le test a réussi.
$TestZero_2(Head : Tail) \rightarrow Test_1(Head.Tail)$

Note: La grammaire produite est de taille polynomiale, mais ce processus est de durée exponentielle en fonction de la taille du compteur (i.e du nombre d'actions empilées), et ses arbres intermédiaires sont également de taille exponentielle. Si l'on veut gagner en efficacité on peut systématiquement remplacer un compteur testé à zéro par $E(B^6E)$ sous la tête de lecture: si le compteur vaut zéro, cela ne change rien. Sinon, la dérivation échouera de toute façon à cause de l'échec du test.

- Le test NonZero est très similaire, à l'exception près qu'après les divisions par 15, on doit pouvoir diviser encore par 3 (témoin d'un Inc) sans diviser par 5 (témoin d'un Dec), au moins une fois.

  On crée Div_3 sur le même modèle que Div_{15}. On l'applique à la fin d'une copie conforme de TestZero (nommée NonZero) et $TestZero_2$ (nommée $NonZero_2$). On l'applique plusieurs fois jusqu'à ce qu'il ne reste plus qu'un seul B. Si on y parvient, le test est réussi.

Ces tests sont générés comme frères de la tête de lecture à chacune des ses transitions:

- $Test_2$ des compteurs est testé à chaque opération;
- Si l'on décrémente un compteur, on le vérifie avec NonZero;
- S'il y a un test x ou y = 0, on vérifie le/les compteurs concerné(s) avec TestZero.

Pour finir, la tête de lecture se supprime ainsi que tous ses fils lorsqu'elle atteint l'état q.

On a donc une dérivation dans cette grammaire G si et seulement si q est accessible dans M.

Cette simulation donne aussi l'indécidabilité du problème d'appartenance.

Théorème III.4. *Soit M une machine de Minski à deux compteurs, q un état de M. On peut calculer une grammaire G Head:Tail non-discriminante et un arbre T tels que G reconnait T $\Leftrightarrow$ q est accessible dans M.*

Démonstration. On utilise la même méthode que pour la démonstration précédente, obtenant une grammaire G qui ne reconnait rien si q est inaccessible dans M mais reconnait ϵ s'il l'est. Ainsi, (G, ϵ) est une instance du problème d'appartenance équivalent à l'accessibilité de q dans M.

L'accessibilité d'un état dans une machine de Minski deux compteurs étant indécidable, on a prouvé par réduction que le vide et l'appartenance sont indécidable pour Head:Tail non-discriminant, et donc pour Head:Tail discriminant (qui simule le précédent).

Ces résultats démontrent que le formalisme Head:Tail qui permet naturellement de différentier deux fils (nécessaire pour décrire $L(G_0)$) et de remplir la première condition est "trop expressif" pour remplir la seconde condition malgré sa simplicité.

Part IV
Restriction Top-Down

Des deux formalismes présentés plus hauts, aucun ne remplit toutes les conditions requises:

- L'approche Unary n'est pas assez expressive pour décrire tous les langages CFTL rang borné.
- Le formalisme Head:Tail est trop expressif pour permettre aux problèmes du vide et d'appartenance d'être décidables.

De plus, on n'a pas trouvé de formalisme d'automates équivalent à Unary, et il ne peut en exister pour Head:Tail (appartenance indécidable).

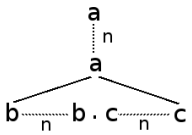
La piste explorée ici est le choix d'une restriction à imposer au formalisme Head:Tail qui rendrait les problèmes du vide et de l'appartenance décidables, et l'équivalence avec un formalisme d'automates possible.
On choisit de forcer la grammaire Head:Tail à dériver de façon top-down car, contrairement à CFTL rang borné et Unary, forcer une grammaire Head:Tail à se dériver de manière top-down change le langage qu'elle reconnait. Cette propriété rend plus difficile la comparaison avec les formalismes d'automates, qui sont top-down par nature.

Exemple. *Soit* $G = (\mathcal{N} = \{S_0, S, N, M\}, S_0, \Delta)$*, où* $\Delta =$

- $S_0(Head : Tail) \to S(N(\epsilon), M(\epsilon))$
- $S(Head : Tail) \to Head$
- $N(Head : Tail) \to \epsilon$
- $M(Head : Tail) \to a$

En forçant la dérivation top-down, on reconnait $\{\epsilon\}$*, alors que la grammaire non-restreinte reconnait* $\{\epsilon, a\}$

Certains résultats passent directement de Head:Tail à Top-down Head:Tail, par exemple la démonstration de CFTL⊂Top-down Head:Tail ou encore le fait que l'on puisse décrire:



On peut également démontrer que Top-Down Head:Tail est plus expressif que $FCNS^{-1}$(CFTL rang borné):

Théorème IV.1. *Soit $G = \{\mathcal{N}, S_0, \Delta\}$ une grammaire CFTL rang borné qui décrit FCNS(L); il existe une grammaire Top-down Head:Tail $G' = \{\mathcal{N}', S_0, \Delta'\}$ qui décrit L.*

Démonstration. Il s'agit encore une fois de déplier l'encodage FCNS au fil de la dérivation. La transformation de Δ qui en résulte sera plus simple que dans le cas général:

- Les dérivations de G seront directement traduites, comme dans la démonstration de CFTL rang borné $\subset$ Head:Tail. Cependant, on remplacera les lettres $a \in \mathcal{F}$ par une copie $U_a \in Unfold_{\mathcal{F}}$ dans les membres droits de Δ, qui sert à s'assurer qu'on déplie correctement l'encodage FCNS une fois la dérivation dans G simulée.
- Chaque $U_a \in Unfold_{\mathcal{F}}$ possède une unique transition qui lui permet de déplier FCNS:
  $U_a(Head : Tail) \rightarrow a(Head).Tail$
  Ainsi, une fois la dérivation simulant G terminée, les copies $U_a \in Unfold_{\mathcal{F}}$déplient le FCNS afin de retrouver L à l'arrivée.

Un chemin de dérivation top-down de G dans CFTL rang borné est donc traduit en un chemin de dérivation top-down "dépliant" dans G', grammaire Top-down Head:Tail. Ainsi G' décrit $FCNS^{-1}(L(G)) = L$

On retrouve donc la première propriété requise, c'est-à-dire que Top-Down Head:Tail étend CFTL rang borné. Cependant, on peut démontrer, comme pour Head:Tail classique, que les problèmes du vide et de l'appartenance sont indécidables. La démonstration passe à nouveau par la simulation d'une machine de Minski à deux compteurs et son problème d'accessibilité.

Théorème IV.2. *Soit une machine de Minski M, un de ses états q. On peut décider d'une grammaire Top-Down Head:Tail telle que son langage est non-vide si et seulement si q est accessible dans M.*

Démonstration. Cette démonstration est basée sur la même structure que la simulation d'une machine de Minski par une grammaire non-discriminante, quoique rendue plus simple par la certitude qu'on dérivera dans l'ordre top-down: on s'y servira d'ailleurs des non-terminaux E effaçables et B bloquants introduits plus haut. Il s'agit de modéliser la tête de lecture, les compteurs et les actions, et les tests.

Première étape: Simuler la tête de lecture

La tête de lecture est modélisée par son état (un non-terminal de la grammaire) et l'étape dans laquelle se trouve l'action en cours (utile uniquement pour décrémenter le deuxième compteur). Les compteurs sont codés dans ses fils. Étant le noeud le plus haut, il est dérivé jusqu'à sa suppression.

Simuler une transition revient à dériver la tête de l'état de départ vers l'état d'arrivée, et à appliquer aux compteurs l'action qui correspond. La tête de lecture génèrera également les tests nécessaires à la validité de la dérivation, au

niveau de ses frères.

Deuxième étape: Simuler les compteurs

Les compteurs sont encodés sous la tête de lecture de la manière suivante: $B^n.E.B^m$, où n est la valeur du premier compteur x, m celle du second compteur y.

Simuler l'incrémentation de x (respectivement y) revient à ajouter un B à gauche (respectivement à droite) de la liste des fils de la tête de lecture.

Simuler la décrémentation revient simplement à supprimer la tête de la liste des fils, à condition de vérifier que la valeur actuelle de x n'est pas 0.
La décrémentation de y est plus complexe, on la décrira après avoir décrit les divers tests nécessaires à son élaboration.

Troisième étape: Contrôler avec des tests

On a besoin de quelques tests pour certifier le bon fonctionnement de la dérivation; afin de les créer on se servira de tests de base présentés ici:

- On se servira des $Test_0$ et $Test_1$ définis plus haut, pour vérifier qu'une haie est de taille 0 ou 1.
- $Test_{\overline{0}}$ est un test créé pour vérifier si la liste de fils donnée est non-vide: $Test_{\overline{0}}(Head : Tail) \rightarrow Test_1(Head)$
- $Test_{>1}$ est un test créé pour vérifier si la liste de fils donnée est de taille 1 ou plus:
  $Test_{>1}(Head : Tail) \rightarrow Test_{\overline{0}}(Tail)$
- TestE : ce test vérifie que ce qu'on lui donne est un simple E:
  $TestE(Head : Tail) \rightarrow Head.Test_0(Tail)$
- ContientE : ce test vérifie qu'il existe un E parmi la liste de fils qu'on lui donne:
  $ContientE(Head : Tail) \rightarrow ContientE(Tail)$ et
  $ContientE(Head : Tail) \rightarrow TestE(Head)$
- $ContientE^2$ est très similaire: il s'agit de déterminer si E^2 est présent dans la liste des fils. $ContientE^2(Head : Tail) \rightarrow ContientE^2(Tail)$ et
  $ContientE^2(Head : Tail) \rightarrow TestE(Head).ContientE'(Tail)$.
  $ContientE'(Head : Tail) \rightarrow TestE(Head)$
- GotoE + Test : GotoE est une structure à utiliser avec un second test: il détecte un E, l'élimine, puis passe le reste des fils à un second test:
  $GotoE_{AutreTest}(Head : Tail) \rightarrow GotoE_{AutreTest}(Tail)$ et
  $GotoE_{AutreTest}(Head : Tail) \rightarrow TestE(Head).AutreTest(Tail)$

On se sert de ces tests afin de construire les tests utiles à la simulation d'une machine de Minski:

- X-Zero teste si la valeur actuelle du compteur x vaut 0:
  $X - Zero(Head : Tail) \rightarrow TestE(Head)$

- $X - \overline{Zero}$ teste au contraire si la valeur actuelle du compteur x est différente de 0:
  $X - \overline{Zero}(Head : Tail) \rightarrow ContientE(Tail)$

- Y-Zero (respectivement Un) teste si la valeur actuelle du compteur y vaut 0 (respectivement 1):
  $Y - Zero(Head : Tail) \rightarrow GotoE_{Test_0}(Head : Tail)$
  $Y - Un(Head : Tail) \rightarrow GotoE_{Test_1}(Head : Tail)$

- $Y - \overline{Zero}$ teste au contraire si la valeur actuelle du compteur y est différente de 0:
  $Y - \overline{Zero}(Head : Tail) \rightarrow GotoE_{Test_{\overline{0}}}(Head : Tail)$

- $Y - SupUn$: teste que la valeur du compteur y est supérieure à 1:
  $Y - SupUn(Head : Tail) \rightarrow GotoE_{Test_{>1}}(Head : Tail)$

Il ne reste plus qu'à élaborer la décrémentation de y. On la découpera en trois cas:

- $x = 0$, $y \neq 0$: Le compteur est de la forme: $E.B^m$. On produit les tests $X - Zero$ et $Y - \overline{Zero}$; le compteur est débarrassé deux fois de sa tête, puis on ajoute un E à gauche:
  $q(Head : Tail) \rightarrow (q, DecY_{x=0})(Head.Tail).X - Zero(Head.Tail).Y - \overline{Zero}(Head.Tail)$
  $(q, DecY_{x=0})(Head : Tail) \rightarrow (q, DecY_{x=0}[1])(Tail)$
  $(q, DecY_{x=0}[1])(Head : Tail) \rightarrow q'(E.Tail)$
  La dérivation des compteurs est donc de la forme:
  $E.B^m \rightarrow B^m \rightarrow E.B^{m-1}$

- $x \neq 0$, $y = 1$: Le compteur est de la forme: $B^n.E.B$. On produit les tests $X - \overline{Zero}$ et $Y - Un$.
  $q(Head : Tail) \rightarrow (q, DecY_{x\neq 0,y=1})(Head.Tail).X - \overline{Zero}(Head.Tail).Y - Un(Head.Tail)$
  On lit le compteur en rajoutant les fils lus en fin de liste, jusqu'à trouver E; on élimine le B qui suit, puis on replace le E en fin de liste:
  $(q, DecY_{x\neq 0,y=1})(Head : Tail) \rightarrow (q, DecY_{x\neq 0,y=1})(Tail.Head)$
  $(q, DecY_{x\neq 0,y=1})(Head : Tail) \rightarrow (q, DecY_{x\neq 0,y=1}[1])(Tail).TestE(Head)$
  $(q, DecY_{x\neq 0,y=1}[1])(Head : Tail) \rightarrow q'(Tail.E)$
  La dérivation des compteurs est donc de la forme:
  $B^n.E.B \rightarrow E.B^{n+1} \rightarrow B^{n+1} \rightarrow B^n.E$

- $x \neq 0$, $y > 1$: Le compteur est de la forme $B^n.E.B^m$. On produit les tests $X - \overline{Zero}$ et $Y - SupUn$.
  $q(Head : Tail) \rightarrow (q, DecY_{x\neq 0,y>1})(Head.Tail).X - \overline{Zero}(Head.Tail).Y - SupUn(Head.Tail)$
  On commence par marquer la fin de y avec le motif EE:

$(q, DecY_{x\neq 0,y>1})(Head:Tail) \to (q, DecY_{x\neq 0,y>1}[1])(Head.Tail.E.E)$
Compteurs: $B^n.E.B^m \to B^n.E.B^m.E^2$

Ensuite, on lit le compteur en rajoutant les fils lus en fin de liste, jusqu'à trouver E; on vérifie que la suite contient le motif E^2, grâce au test $ContientE^2$:
$(q, DecY_{x\neq 0,y>1}[1])(Head:Tail) \to (q, DecY_{x\neq 0,y>1}[1])(Tail.Head)$
$(q, DecY_{x\neq 0,y>1}[1])(Head:Tail) \to (q, DecY_{x\neq 0,y>1}[2])(Tail).TestE(Head).ContientE^2(Tail)$
Compteurs: $B^n.E.B^m.E^2 \to B^m.E^2.B^n$

Après cela, on retire un B, on rajoute le E en fin de liste:
$(q, DecY_{x\neq 0,y>1}[2])(Head:Tail) \to (q, DecY_{x\neq 0,y>1}[3])(Tail.E)$
Compteurs: $B^m.E^2.B^n \to B^{m-1}.E^2.B^n.E$

Enfin, on lit le compteur en rajoutant les fils lus en fin de liste, jusqu'à trouver E^2; on l'élimine et l'on obtient le compteur décrémenté:
$(q, DecY_{x\neq 0,y>1}[3])(Head:Tail) \to (q, DecY_{x\neq 0,y>1}[3])(Tail.Head)$
$(q, DecY_{x\neq 0,y>1}[3])(Head:Tail) \to (q, DecY_{x\neq 0,y>1}[4])(Tail).TestE(Head)$
$(q, DecY_{x\neq 0,y>1}[4])(Head:Tail) \to q'(Tail).TestE(Head)$
Compteurs: $B^{m-1}.E^2.B^n.E \to B^n.E.B^{m-1}$

Les tests ne se dérivent en ϵ que s'ils réussissent. Dans le cas contraire, le test devient bloquant.

Quant à la tête de lecture, elle se dérive en ϵ lorsqu'elle atteint l'état q qu'on étudie.

La grammaire ainsi construite reconnait ϵ si et seulement si q est accessible dans M.

Cette simulation donne aussi l'indécidabilité du problème d'appartenance.

Théorème IV.3. *Soit M une machine de Minski à deux compteurs, q un état de M. On peut calculer une grammaire G Head:Tail non-discriminante et un arbre T tels que G reconnait T $\Leftrightarrow$ q est accessible dans M.*

Démonstration. On utilise la même méthode que pour la démonstration précédente, obtenant une grammaire G qui ne reconnait rien si q est inaccessible dans M mais reconnait ϵ s'il l'est. Ainsi, (G, ϵ) est une instance du problème d'appartenance équivalent à l'accessibilité de q dans M.

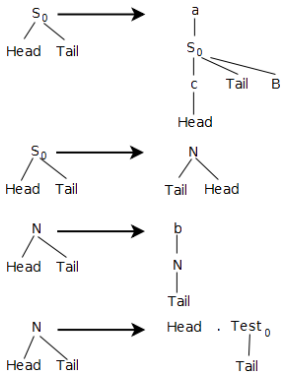
Conclusion

Les formalismes présentés dans ce rapport ne possèdent pas toutes les conditions que l'on désire pour un formalisme CFTL sur les arbres sans rang: le formalisme Unary par manque d'expressivité, et Head:Tail pour être trop expressif. Restreindre les grammaires Head:Tail à une dérivation top-down ne suffit pas à limiter suffisamment cette expressivité.

On pourrait essayer de continuer à restreindre le formalisme Head:Tail en forçant une autre propriété que l'on perd en passant des grammaires rang borné aux arbres sans rang.

On remarque que les démonstrations d'indécidabilité des problèmes du vide et de l'appartenance reposent sur la capacité du formalisme à compter les fils, fut-ce sous certaines conditions. Cela semble assez directement opposé à l'idée de Context-Free. Par exemple, on peut utiliser cette propriété pour reconnaitre le langage des arbres $a^n b^n c^n$ verticaux:

En plus de ses deux non-terminaux S_0 en N, cette grammaire utilise $Test_0$ et B définis dans ce rapport.



Afin d'éviter cet effet de comptage, une piste serait d'interdire la supression de fils. Ils n'est pas certain que toute grammaire rang borné soit équivalente à une grammaire sans suppression de fils, i.e qui utilise tous les fils d'un noeud à chaque dérivation. Si c'est le cas, on pourra essayer d'imposer cette propriété à Head:Tail (Top-Down ou non) et observer les propriétés de ce nouveau formalisme.

Par ailleurs, plusieurs questions deumeurent ouvertes quant à ces formalismes, comme l'existence d'une classe d'automates équivalente à Unary, l'égalité/inégalité de Head:Tail discriminant ou non, et de ceux-ci à Head:Tail Top-Down, qui n'est pas forcément plus faible que le cas général.