Langages et Automates : LA3
Partie 7 : Grammaires

Introduction à la notion de Grammaire

Grammaire = mécanisme de génération de langages

Introduction à la notion de Grammaire

Grammaire = mécanisme de génération de langages

Plus général que les expressions réguières
ex : on pourra exprimer $\{a^n b^n,\ n \in \mathbb{N}\}$

Introduction à la notion de Grammaire

Grammaire = mécanisme de génération de langages

Plus général que les expressions réguières
ex : on pourra exprimer $\{a^n b^n,\ n \in \mathbb{N}\}$

Introduites notamment par Noam Chomsky pour formaliser les propriétés grammaticales des langues naturelles (traduction automatique)

Introduction à la notion de Grammaire

Grammaire = mécanisme de génération de langages

Plus général que les expressions réguières
ex : on pourra exprimer $\{a^n b^n, n \in \mathbb{N}\}$

Introduites notamment par Noam Chomsky pour formaliser les propriétés grammaticales des langues naturelles (traduction automatique)

Elles ont un grand intérêt aussi en informatique, pour les notions de calculabilité, de complexité, ou pour la compilation de programmes.

Introduction à la notion de Grammaire

En programmation, une étape cruciale est la phase de *compilation*,

Introduction à la notion de Grammaire

En programmation, une étape cruciale est la phase de *compilation*,

Compilation = la traduction d'un programme d'un langage A (typiquement le langage de programmation, compréhensible par des humains) vers un langage B (typiquement un binaire compréhensible par l'ordinateur).

Introduction à la notion de Grammaire

En programmation, une étape cruciale est la phase de *compilation*,

Compilation = la traduction d'un programme d'un langage A (typiquement le langage de programmation, compréhensible par des humains) vers un langage B (typiquement un binaire compréhensible par l'ordinateur).

Première étape de la compilation : vérifier par un algorithme si le lexique (les mots utilisés) et la syntaxe (la structure des phrases) est correcte.

Introduction à la notion de Grammaire

En programmation, une étape cruciale est la phase de *compilation*,

Compilation = la traduction d'un programme d'un langage A (typiquement le langage de programmation, compréhensible par des humains) vers un langage B (typiquement un binaire compréhensible par l'ordinateur).

Première étape de la compilation : vérifier par un algorithme si le lexique (les mots utilisés) et la syntaxe (la structure des phrases) est correcte.

Pour vérifier la syntaxe, il faut donc spécifier et décrire la grammaire du langage A, c'est à dire la façon dont peuvent être construites les phrases autorisées.

Exemple

Avant de définir rigoureusement une grammaire, disons qu'il s'agit d'un ensemble de règles de substitution permettant de produire des phrases à partir d'un symbole de départ (souvent noté S).

Exemple

Avant de définir rigoureusement une grammaire, disons qu'il s'agit d'un ensemble de règles de substitution permettant de produire des phrases à partir d'un symbole de départ (souvent noté S).

Par exemple pour la langue française, une grammaire très sommaire pourrait être :

- S := GN GV
- GN := Ar Ad N
- GN := GN PP GN
- GV := V
- GV := V GN

(La barre verticale | signifie juste OU)

- Ar := le | la | un | une | ε
- Ad := petit | grand | jeune | ε
- V := regarde | mange
- PP := de | à
- N := Pierre | Toto | train |fille | viande | dessert

Exemple

Pour générer la phrase "La jeune fille de pierre mange", il suffit d'appliquer une suite de substitutions suivante :

$$
\begin{aligned}
S &:= GN\ GV \\
&:= GN\ PP\ GN\ GV \\
&:= Ar\ Ad\ N\ PP\ GN\ GV \\
&:= Ar\ N\ PP\ GN\ V \\
&:= la\ N\ PP\ GN\ V \\
&:= la\ fille\ PP\ GN\ V \\
&:= la\ fille\ de\ GN\ V \\
&:= la\ fille\ de\ Ar\ Ad\ N\ V \\
&:= la\ fille\ de\ Pierre\ V \\
&:= la\ fille\ de\ Pierre\ mange
\end{aligned}
$$

Exemple

Pour générer la phrase "Le petit viande mange un jeune train", il suffit d'appliquer une suite de substitutions suivante :

$$
\begin{aligned}
S &:= GN\ GV \\
&:= Ar\ Ad\ N\ GV \\
&:= Ar\ Ad\ N\ V\ GN \\
&:= Ar\ Ad\ N\ V\ Ar\ Ad\ N \\
&:= le\ Ad\ N\ V\ Ar\ Ad\ N \\
&:= le\ petit\ N\ V\ Ar\ Ad\ N \\
&:= le\ petit\ viande\ V\ Ar\ Ad\ N \\
&:= le\ petit\ viande\ V\ Ar\ Ad\ train \\
&:= le\ petit\ viande\ V\ un\ Ad\ train \\
&:= le\ petit\ viande\ mange\ un\ Ad\ train \\
&:= le\ petit\ viande\ mange\ un\ jeune\ train
\end{aligned}
$$

Exemple

Comme ce qui nous interesse c'est la structure des phrases (la grammaire) et pas au lexique (quels mots sont autorisées ou pas), en fait les mots "le", "la", "Pierre", "toto" seront en fait représentées par des symboles : des lettres.

- S := GN GV
- GN := Ar Ad N
- GN := GN PP GN
- GV := V
- GV := V GN

- Ar := le | la | un | une
- Ad := petit | grand | jeune | ε
- V := regarde | mange
- PP := de | à
- N := Pierre | Toto | train |fille | viande | dessert

Exemple

Comme ce qui nous interesse c'est la structure des phrases (la grammaire) et pas au lexique (quels mots sont autorisées ou pas), en fait les mots "le", "la", "Pierre", "toto" seront en fait représentées par des symboles : des lettres.

- S := GN GV
- GN := Ar Ad N
- GN := GN PP GN
- GV := V
- GV := V GN

- Ar := a | b | c | d | ε
- Ad := e | f | g | ε
- V := i | j
- PP := k | l
- N := m | n | o | p | q | r

Exemple

Comme ce qui nous interesse c'est la structure des phrases (la grammaire) et pas au lexique (quels mots sont autorisées ou pas), en fait les mots "le", "la", "Pierre", "toto" seront en fait représentées par des symboles : des lettres.

- $S := V_1 \ V_2$
- $V_1 := V_3 \ V_4 \ V_5$
- $V_1 := V_1 \ V_6 \ V_1$
- $V_2 := V_7$
- $V_2 := V_7 \ V_1$

- $V_3 :=$ a | b | c | d | ε
- $V_4 :=$ e | f | g | ε
- $V_7 :=$ i | j
- $V_6 :=$ k | l
- $V_5 :=$ m | n | o | p | q | r

Exemple

Comme ce qui nous interesse c'est la structure des phrases (la grammaire) et pas au lexique (quels mots sont autorisées ou pas), en fait les mots "le", "la", "Pierre", "toto" seront en fait représentées par des symboles : des lettres.

- $S := V_1 \ V_2$
- $V_1 := V_3 \ V_4 \ V_5$
- $V_1 := V_1 \ V_6 \ V_1$
- $V_2 := V_7$
- $V_2 := V_7 \ V_1$
- $V_3 := \mathsf{a} \mid \mathsf{b} \mid \mathsf{c} \mid \mathsf{d} \mid \varepsilon$
- $V_4 := \mathsf{e} \mid \mathsf{f} \mid \mathsf{g} \mid \varepsilon$
- $V_7 := \mathsf{i} \mid \mathsf{j}$
- $V_6 := \mathsf{k} \mid \mathsf{l}$
- $V_5 := \mathsf{m} \mid \mathsf{n} \mid \mathsf{o} \mid \mathsf{p} \mid \mathsf{q} \mid \mathsf{r}$

Une phrase correcte ne contiendra plus de symboles rouges et sera produite a partir de ces règles de substitution.

Grammaire

Un autre exemple pourrait être la structure des instructions dans un langage

- Identificateur := ....
- Expression := Expression + Expression
- ...
- Instruction := Identificateur = Expression ;
- Instruction := InstructionConditio
- InstructionConditio := if(Condition) then Instruction else Instruction
- InstructionConditio := while(Condition) then Instruction

Définition

Une grammaire algébrique (ou hors contexte) est définie par un quadruplet $G = (\Sigma, V, S, P)$ où

- *Σ est un ensemble de symboles dits symboles terminaux*
- *V est un ensemble (disjoint de Σ) de symboles dits symboles non terminaux*
- *$S \in V$ est un symbole non terminal particulier appelé axiome*
- *$P \subset (V \times (\Sigma \cup V)^*)$ est l'ensemble des règles de production*

Définition

Une grammaire algébrique (ou hors contexte) est définie par un quadruplet $G = (\Sigma, V, S, P)$ où

- *Σ est un ensemble de symboles dits symboles terminaux*
- *V est un ensemble (disjoint de Σ) de symboles dits symboles non terminaux*
- *$S \in V$ est un symbole non terminal particulier appelé axiome*
- *$P \subset (V \times (\Sigma \cup V)^*)$ est l'ensemble des règles de production*

Une règle $(u, v) \in P$ s'écrira $u \to v$ et signifie "on peut substituer u par v".

Définition

Une grammaire algébrique (ou hors contexte) est définie par un quadruplet $G = (\Sigma, V, S, P)$ où

- *Σ est un ensemble de symboles dits symboles terminaux*
- *V est un ensemble (disjoint de Σ) de symboles dits symboles non terminaux*
- *$S \in V$ est un symbole non terminal particulier appelé axiome*
- *$P \subset (V \times (\Sigma \cup V)^*)$ est l'ensemble des règles de production*

Une règle $(u, v) \in P$ s'écrira $u \to v$ et signifie "on peut substituer u par v".

On utilisera par convention des lettres minuscules pour les symboles terminaux et majuscules pour les non terminaux.

Définition

Une grammaire algébrique (ou hors contexte) est définie par un quadruplet $G = (\Sigma, V, S, P)$ où

- *Σ est un ensemble de symboles dits symboles terminaux*
- *V est un ensemble (disjoint de Σ) de symboles dits symboles non terminaux*
- *$S \in V$ est un symbole non terminal particulier appelé axiome*
- *$P \subset (V \times (\Sigma \cup V)^*)$ est l'ensemble des règles de production*

Une règle $(u, v) \in P$ s'écrira $u \to v$ et signifie "on peut substituer u par v".

On utilisera par convention des lettres minuscules pour les symboles terminaux et majuscules pour les non terminaux.

Si plusieurs règles ont le même membre gauche, par exemple $u \to v$, $u \to w$, on utilisera la notation $u \to v|w$.

Grammaire - Exemples

Considérons l'exemple $G_1 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- les règles de production P données par

  $S \to abS$
  $S \to \varepsilon$

On peut prouver que le langage généré est exactement ? ? ?

Grammaire - Exemples

Considérons l'exemple $G_1 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- les règles de production P données par

  $S \to abS$
  $S \to \varepsilon$

On peut prouver que le langage généré est exactement $\{(ab)^n,\ n \in \mathbb{N}\}$.

Grammaire - Exemples

Considérons l'exemple $G_2 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- les règles de production P données par

  $S \to aSb \mid \varepsilon$

On peut prouver que le langage généré est

Considérons l'exemple $G_2 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- les règles de production P données par

  $S \to aSb \mid \varepsilon$

On peut prouver que le langage généré est $\{a^n b^n,\ n \in \mathbb{N}\}$.

Grammaire - Exemples

Considérons l'exemple $G_3 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- les règles de production P données par

$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow a \mid b \mid \varepsilon$$

Le langage généré est :

Grammaire - Exemples

Considérons l'exemple $G_3 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S\}$
- les règles de production P données par

$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow a \mid b \mid \varepsilon$$

Le langage généré est : les palindromes.

Considérons l'exemple $G_4 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X\}$
- les règles de production P données par
  - $S \to aX \mid SS \mid bSaa \mid \varepsilon$
  - $X \to SX \mid bSaS$

Le langage généré est

Grammaire - Exemples

Considérons l'exemple $G_4 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X\}$
- les règles de production P données par
  - $S \rightarrow aX \mid SS \mid bSaa \mid \varepsilon$
  - $X \rightarrow SX \mid bSaS$

Le langage généré est l'ensemble des mots w tels que $|w|_a = 2|w|_b$

Grammaire - Exemples

Considérons l'exemple $G_4 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X\}$
- les règles de production P données par
  - $S \to aX \mid SS \mid bSaa \mid \varepsilon$
  - $X \to SX \mid bSaS$

Le langage généré est l'ensemble des mots w tels que $|w|_a = 2|w|_b$

On peut aussi générer ce langage a l'aide des règles :

$$S \to SS \mid SbS \mid bSaa \mid \varepsilon$$

Grammaire - Exemples

Considérons l'exemple $G_4 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X\}$
- les règles de production P données par
  - $S \rightarrow aX \mid SS \mid bSaa \mid \varepsilon$
  - $X \rightarrow SX \mid bSaS$

Le langage généré est l'ensemble des mots w tels que $|w|_a = 2|w|_b$

On peut aussi générer ce langage a l'aide des règles :

$$S \rightarrow SS \mid SbS \mid bSaa \mid \varepsilon$$

Remarque

Deux grammaires distinctes peuvent générer le même langage. On dit alors que les grammaires sont équivalentes.

Grammaire

Pour certains langages on peut avoir besoin de grammaires plus générales que les grammaires hors contexte (ou algébriques)

ms

On peut pouvoir avoir des règles du type $aX \rightarrow aa$. (on peut remplacer X par un a que si X est deja précédé d'un a).

Grammaire

Pour certains langages on peut avoir besoin de grammaires plus générales que les grammaires hors contexte (ou algébriques)
ms
On peut pouvoir avoir des règles du type $aX \rightarrow aa$. (on peut remplacer X par un a que si X est deja précédé d'un a).

On dit que ces règles sont contextuelles, elles dépendent du contexte dans lequel on se trouve.

Pour certains langages on peut avoir besoin de grammaires plus générales que les grammaires hors contexte (ou algébriques)
ms
On peut pouvoir avoir des règles du type $aX \rightarrow aa$. (on peut remplacer X par un a que si X est deja précédé d'un a).

On dit que ces règles sont contextuelles, elles dépendent du contexte dans lequel on se trouve.

Définition

Une grammaire formelle est définie par un quadruplet $G = (\Sigma, V, S, P)$ *où*

- Σ *est un ensemble de symboles dits symboles terminaux*
- V *est un ensemble (disjoint de* Σ*) de symboles dits symboles non terminaux*
- $S \in V$ *est un symbole non terminal particulier appelé axiome*
- $P \subset ((\Sigma \cup V)^{+} \times (\Sigma \cup V)^{*})$ *est l'ensemble des règles de production*

Grammaire - Exemples

Considérons l'exemple $G_5 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X, Y\}$
- les règles de production P données par

$$
\begin{aligned}
S &\to aSXa \\
S &\to aba \\
aX &\to Xa \\
bX &\to bb
\end{aligned}
$$

On peut prouver que le langage généré est exactement

Grammaire - Exemples

Considérons l'exemple $G_5 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X, Y\}$
- les règles de production P données par

$$S \rightarrow aSXa$$
$$S \rightarrow aba$$
$$aX \rightarrow Xa$$
$$bX \rightarrow bb$$

On peut prouver que le langage généré est exactement $\{a^n b^n a^n,\ n \in \mathbb{N}\}$.

Dérivation

On formalise mathématiquement ce dont nous avons parlé précédemment.

Définition

- *Si $G = (\Sigma, V, S, P)$ est une grammaire et u et v deux mots de $(V \cup \Sigma)^*$, on dit que G permet de dériver v à partir de u en une étape, que l'on note $u \Rightarrow_G v$, si il existe x, y, u', et v' dans $(V \cup \Sigma)^*$ tels que*
  - $u = xu'y$
  - $v = xv'y$
  - *$u' \to v'$ est dans P*
- *G permet de dériver v à partir de u en plusieurs étapes, noté $u \overset{*}{\Rightarrow}_G v$ si il existe $u_0, \ldots u_k$ dans $(V \cup \Sigma)^*$ tels que*
  - $u = u_0$
  - $v = v_k$
  - *Pour tout $i < k$, G permet de dériver u_{i+1} à partir de u_i en une étape.*

(Si il n'y a pas d'ambiguité, et afin d'alléger les notations, on écrira souvent $u \to v$ et $u \Rightarrow v$ au lieu de $u \Rightarrow_G v$ et $u \overset{*}{\Rightarrow}_G v$)

Langage généré

Définition

- *Le langage généré par une grammaire* G *d'axiome* S *est l'ensemble de tous les mots* $u \in \Sigma^*$ *(que des symboles terminaux) tels que* $S \Rightarrow u$*. On le notera* $L(G)$*.*
- *Un langage* $L \subset \Sigma^*$ *tel qu'il existe une grammaire* L *avec* $L = L(G)$ *est dit récursivement énumérable.*
- *Un langage* L *est dit algébrique (ou hors contexte) si il existe une grammaire algébrique qui le génère.*

Langage généré - Remarques

Lorsque l'on veut prouver qu'une grammaire G génère un langage L comme dans l'exemple ci dessus, il faut montrer deux choses :

- $L(G) \subset L$: tout mot généré par G apaprtient à L
- $L \subset L(G)$: tout mot de L peut être obtenu par les règles de production de G

Langage généré - Remarques

Lorsque l'on veut prouver qu'une grammaire G génère un langage L comme dans l'exemple ci dessus, il faut montrer deux choses :

- $L(G) \subset L$: tout mot généré par G apaprtient à L
- $L \subset L(G)$: tout mot de L peut être obtenu par les règles de production de G

Deux grammaires distinctes pouvant générer le même langage, il se peut qu'un langage algébrique soit généré par une grammaire qui ne l'est pas. Pour montrer qu'un langage n'est pas algébrique, il faut bien montrer que toute grammaire qui le génère n'est pas algébrique.

Grammaire - Exemples

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

Grammaire - Exemples

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

1) Prouvons que $L \subset L(G)$ par recurrence sur la longueur des mots de L.

Grammaire - Exemples

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

1) Prouvons que $L \subset L(G)$ par recurrence sur la longueur des mots de L.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans L.

Grammaire - Exemples

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

1) Prouvons que $L \subset L(G)$ par recurrence sur la longueur des mots de L.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans L.
- Induction : Si u de longueur $n \geq 2$ palindrome, alors $u = ava$ ou $u = bvb$ où v est un palindrome plus court.
  Par hypothèse de récurrence $v \in L(G)$ donc $S \overset{*}{\Rightarrow} v$.
  Mais alors $S \Rightarrow aSa \overset{*}{\Rightarrow} ava$ et de même $S \overset{*}{\Rightarrow} bvb$ et donc $S \overset{*}{\Rightarrow} u$, c.a.d $u \in L(G)$.

Grammaire - Exemples

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

1) Prouvons que $L \subset L(G)$ par recurrence sur la longueur des mots de L.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans L.
- Induction : Si u de longueur $n \geq 2$ palindrome, alors $u = ava$ ou $u = bvb$ où v est un palindrome plus court.
  Par hypothèse de récurrence $v \in L(G)$ donc $S \overset{*}{\Rightarrow} v$.
  Mais alors $S \Rightarrow aSa \overset{*}{\Rightarrow} ava$ et de même $S \overset{*}{\Rightarrow} bvb$ et donc $S \overset{*}{\Rightarrow} u$, c.a.d $u \in L(G)$.

2) Prouvons que $L(G) \subset L$ par recurrence sur la longueur des mots de $L(G)$.

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

1) Prouvons que $L \subset L(G)$ par recurrence sur la longueur des mots de L.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans L.
- Induction : Si u de longueur $n \geq 2$ palindrome, alors $u = ava$ ou $u = bvb$ où v est un palindrome plus court.
  Par hypothèse de récurrence $v \in L(G)$ donc $S \overset{*}{\Rightarrow} v$.
  Mais alors $S \Rightarrow aSa \overset{*}{\Rightarrow} ava$ et de même $S \overset{*}{\Rightarrow} bvb$ et donc $S \overset{*}{\Rightarrow} u$, c.a.d $u \in L(G)$.

2) Prouvons que $L(G) \subset L$ par recurrence sur la longueur des mots de $L(G)$.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans $L(G)$.

Grammaire - Exemples

Reprenons l'exemple de la grammaire $G = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$, $V = \{S\}$
- les règles $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Soit $L =$ l'ensemble des palindromes (cad tq $u = miroir(u)$).

1) Prouvons que $L \subset L(G)$ par recurrence sur la longueur des mots de L.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans L.
- Induction : Si u de longueur $n \geq 2$ palindrome, alors $u = ava$ ou $u = bvb$ où v est un palindrome plus court.
  Par hypothèse de récurrence $v \in L(G)$ donc $S \overset{*}{\Rightarrow} v$.
  Mais alors $S \Rightarrow aSa \overset{*}{\Rightarrow} ava$ et de même $S \overset{*}{\Rightarrow} bvb$ et donc $S \overset{*}{\Rightarrow} u$, c.a.d $u \in L(G)$.

2) Prouvons que $L(G) \subset L$ par recurrence sur la longueur des mots de $L(G)$.

- Intialisiation longueur 0 ou 1. Les mots ε, a, n sont tous dans $L(G)$.
- Induction : Si u de longueur $n \geq 2$ est dans $L(G)$, alors la dérivation qui mène à u commence nécessairement par $S \Rightarrow aSa$ ou $S \Rightarrow bSb$. Ce qui signifie que $u = ava$ ou $u = bvb$ où v est tel que $S \overset{*}{\Rightarrow} v$.
  $v \in L(G)$ et $|v| < |u|$, donc par hypothèse de récurrence, v palindrome, donc u palindrome.

Considérons l'exemple $G_6 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b, +, -, /, \times\}$
- $V = \{S\}$
- les règles de production P données par

$$
\begin{array}{ll}
S \rightarrow x|y|z & S \rightarrow S \times S \\
S \rightarrow S + S & S \rightarrow S/S \\
S \rightarrow S - S & S \rightarrow (S)
\end{array}
$$

On peut alors fabriquer toutes les expressions arithmetiques correctment parenthésées sur les variables x, y, z.

Grammaire - Exemples

Considérons l'exemple $G_7 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b, +, ., ^*\}$
- $V = \{S\}$
- les règles de production P données par

$$S \to \varepsilon \qquad S \to S.S$$
$$S \to (S) \qquad S \to S^*$$
$$S \to S + S \qquad S \to a|b$$

On peut alors fabriquer toutes les expressions rationnelles.

Grammaire - Exemples

Considérons l'exemple $G_8 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b, c\}$
- $V = \{S, X\}$
- les règles de production P données par
  - $S \to aXSc$
  - $S \to abc$
  - $Xa \to aX$
  - $Xb \to bb$

Le langage généré est

Grammaire - Exemples

Considérons l'exemple $G_8 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b, c\}$
- $V = \{S, X\}$
- les règles de production P données par
  - $S \to aXSc$
  - $S \to abc$
  - $Xa \to aX$
  - $Xb \to bb$

Le langage généré est $L = \{a^n b^n c^n \;\; n \in \mathbb{N}\}$.

Grammaire - Exemples

Considérons l'exemple $G_9 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b, c, d, ...z, 0, 1, ...9, \}$
- $V = \{< ident >, < lettre >, < chiffre >, < suiteCar >\}$
- $S =< ident >$
- les règles de production P données par :
  - $< ident > \to < lettre >< suiteCar >$
  - $< suiteCar > \to \varepsilon \mid < lettre >< suiteCar > \mid < chiffre >< suiteChar >$
  - $< lettre > \to a|b|c|...|z$
  - $< chiffre > \to 0|1|2...|8|9$

Le langage généré est

Grammaire - Exemples

Considérons l'exemple $G_9 = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b, c, d, ...z, 0, 1, ...9, \}$
- $V = \{< ident >, < lettre >, < chiffre >, < suiteCar >\}$
- $S =< ident >$
- les règles de production P données par :
  - $< ident > \rightarrow < lettre >< suiteCar >$
  - $< suiteCar > \rightarrow \varepsilon \mid < lettre >< suiteCar > \mid < chiffre >< suiteChar >$
  - $< lettre > \rightarrow a|b|c|...|z$
  - $< chiffre > \rightarrow 0|1|2...|8|9$

Le langage généré est l'ensemble des mots qui ne commencent pas par un chiffre (les noms de variables valides en Java par exemple)

Grammaire - Exemples

Considérons l'exemple $G_{10} = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X, Y\}$
- les règles de production P données par
  - $S \rightarrow aSa \mid bSb \mid aTb \mid bTa$
  - $T \rightarrow aT \mid bT \mid \varepsilon$

Le langage généré est l'ensemble des mots non palindromes

Grammaire - Exemples

Considérons l'exemple $G_{11} = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X, Y, T\}$
- les règles de production P données par
  - $S \to aXS \mid bYS \mid T$
  - $Xa \to aX$
  - $Xb \to bX$
  - $Ya \to aY$
  - $Yb \to bY$
  - $YT \to Tb$
  - $XT \to Ta$
  - $T \to \varepsilon$

Le langage généré est

Grammaire - Exemples

Considérons l'exemple $G_{11} = (\Sigma, V, S, P)$ avec

- $\Sigma = \{a, b\}$
- $V = \{S, X, Y, T\}$
- les règles de production P données par
  - $S \to aXS \mid bYS \mid T$
  - $Xa \to aX$
  - $Xb \to bX$
  - $Ya \to aY$
  - $Yb \to bY$
  - $YT \to Tb$
  - $XT \to Ta$
  - $T \to \varepsilon$

Le langage généré est $L = \{uu,\ u \in \Sigma^*\}$.

Grammaires régulières

Les grammaires régulières constituent une sous famille des grammaires algébriques, dont les règles sont simples.

Définition

Une grammaire est dite régulière (ou linéaire à droite) est une grammaire telle que les règles de production sont de la forme :

$$\begin{array}{rll} & A \rightarrow aB & A, B \in V \quad (nonterminaux) \\ ou & A \rightarrow \varepsilon & \end{array}$$

Grammaires régulières

Les grammaires régulières constituent une sous famille des grammaires algébriques, dont les règles sont simples.

Définition

Une grammaire est dite régulière (ou linéaire à droite) est une grammaire telle que les règles de production sont de la forme :

$$\begin{array}{ll} & A \rightarrow aB \quad A, B \in V \quad (nonterminaux) \\ ou & A \rightarrow \varepsilon \end{array}$$

Quels sont les langages correspondants ? ?

Grammaires régulières

Théoreme

Un langage est rationnel si et seulement si il est généré par une grammaire régulière.

Les symboles non terminaux correspondent exactement aux états de l'automate (S étant l'état initial) et les règles de production aux transitions. Pour toute règle $A \to \varepsilon$, l'état A est un état acceptant.

Grammaires régulières

Théoreme

Un langage est rationnel si et seulement si il est généré par une grammaire régulière.

Les symboles non terminaux correspondent exactement aux états de l'automate (S étant l'état initial) et les règles de production aux transitions. Pour toute règle $A \to \varepsilon$, l'état A est un état acceptant.

Formellement : Pour la grammaire, $G = (\Sigma, V, S, P)$ on construit l'automate $\mathcal{A} = (\Sigma, V, S, F, \delta)$, où :

$$F = \{A \in V, P \text{ contient la règle de production } A \to \varepsilon\}$$
$$\delta(A, a) = B \text{ si et seulement si } P \text{ contient la règle de production } A \to aB$$

Théoreme

Un langage est rationnel si et seulement si il est généré par une grammaire régulière.

Les symboles non terminaux correspondent exactement aux états de l'automate (S étant l'état initial) et les règles de production aux transitions. Pour toute règle $A \to \varepsilon$, l'état A est un état acceptant.

Formellement : Pour la grammaire, $G = (\Sigma, V, S, P)$ on construit l'automate $\mathcal{A} = (\Sigma, V, S, F, \delta)$, où :

$$F = \{A \in V, P \text{ contient la règle de production } A \to \varepsilon\}$$

$$\delta(A, a) = B \text{ si et seulement si } P \text{ contient la règle de production } A \to aB$$

La réciproque consiste à faire la même dans le sens inverse (automate vers grammaire). Cela correspond a la construction du système d'équations linéaires gauche (voir chapitre automates).

Hierarichie de Chomsky

Chomsky a défini une hiérarchie de complexité de langages par

- type 0 : non contraintes (langages récursivement énumérables)
- type 1 : contextuelles : $uXv \to uwv$ avec $w \in V^+$
- type 2 : hors-contexte ou algébriques
- type 3 : régulieres

Dans le type 1, le fait important est que le membre droit d'une règle est plus long que le membre gauche. On dit parfois qu'elles sont monotones.

Hierarichie de Chomsky

Chomsky a défini une hiérarchie de complexité de langages par

- type 0 : non contraintes (langages récursivement énumérables)
- type 1 : contextuelles : $uXv \to uwv$ avec $w \in V^+$
- type 2 : hors-contexte ou algébriques
- type 3 : régulieres

Dans le type 1, le fait important est que le membre droit d'une règle est plus long que le membre gauche. On dit parfois qu'elles sont monotones.

En terme de machines, le type 3 correspond au AFD, le type 2 aux automates à pile (cf cours suivant) et les types 0 et 1 a des machines de Turing (avec une contrainte pour différencier les deux types).

Hierarichie de Chomsky

Chomsky a défini une hiérarchie de complexité de langages par

- type 0 : non contraintes (langages récursivement énumérables)
- type 1 : contextuelles : $uXv \rightarrow uwv$ avec $w \in V^+$
- type 2 : hors-contexte ou algébriques
- type 3 : régulieres

Dans le type 1, le fait important est que le membre droit d'une règle est plus long que le membre gauche. On dit parfois qu'elles sont monotones.

En terme de machines, le type 3 correspond au AFD, le type 2 aux automates à pile (cf cours suivant) et les types 0 et 1 a des machines de Turing (avec une contrainte pour différencier les deux types).

A partir de maintenant, on ne considèrera plus que les langages et grammaires algébriques.

Propriétés de Cloture

Théoreme

Les langages algébriques sont clos pour l'union, la concaténation et l'étoile de Kleene.

Théoreme

Les langages algébriques sont clos pour l'union, la concaténation et l'étoile de Kleene.

Si on a deux grammaires G_1 et G_2 de symboles initiaux S_1 et S_2, il suffit de rajouter un symbole initial S et de rajouter une règle $S \to S_1 \mid S_2$ pour obtenir l'union des langages engendrés.

Propriétés de Cloture

Théoreme

Les langages algébriques sont clos pour l'union, la concaténation et l'étoile de Kleene.

Si on a deux grammaires G_1 et G_2 de symboles initiaux S_1 et S_2, il suffit de rajouter un symbole initial S et de rajouter une règle $S \to S_1 \mid S_2$ pour obtenir l'union des langages engendrés.
On fera de même en rajoutante la règle $S \to S_1.S_2$ pour obtenir le produit.

Propriétés de Cloture

Théoreme

Les langages algébriques sont clos pour l'union, la concaténation et l'étoile de Kleene.

Si on a deux grammaires G_1 et G_2 de symboles initiaux S_1 et S_2, il suffit de rajouter un symbole initial S et de rajouter une règle $S \to S_1 \mid S_2$ pour obtenir l'union des langages engendrés.
On fera de même en rajoutante la règle $S \to S_1.S_2$ pour obtenir le produit.
Pour l'étoile, il suffit de rajouter la règle $S \to S_1 S \mid \varepsilon$

Propriétés de Cloture

Théoreme

Les langages algébriques sont clos pour l'union, la concaténation et l'étoile de Kleene.

Si on a deux grammaires G_1 et G_2 de symboles initiaux S_1 et S_2, il suffit de rajouter un symbole initial S et de rajouter une règle $S \to S_1 \mid S_2$ pour obtenir l'union des langages engendrés.
On fera de même en rajoutante la règle $S \to S_1.S_2$ pour obtenir le produit.
Pour l'étoile, il suffit de rajouter la règle $S \to S_1 S \mid \varepsilon$

Proposition

Les langages algébriques NE sont PAS clos pour l'intersection et le complémentaire

Exemple : $L_1 = \{a^n b^n c^m, (n, m) \in \mathbb{N}^2\}$ et $L_2 = \{a^n b^m c^m, (n, m) \in \mathbb{N}^2\}$ sont algébriques (pourquoi ?), mais leur intersection ne l'est pas (voir plus tard).

Arbre de dérivation

Pour une grammaire algébrique $G = (\Sigma, V, S, P)$ un arbre de dérivation pour un mot w à partir du symbole S est un arbre enraciné tel que :

- la racine est étiquetée par S

Arbre de dérivation

Pour une grammaire algébrique $G = (\Sigma, V, S, P)$ un arbre de dérivation pour un mot w à partir du symbole S est un arbre enraciné tel que :

- la racine est étiquetée par S
- Les feuilles sont étiquetées par des symboles terminaux.

Arbre de dérivation

Pour une grammaire algébrique $G = (\Sigma, V, S, P)$ un arbre de dérivation pour un mot w à partir du symbole S est un arbre enraciné tel que :

- la racine est étiquetée par S
- Les feuilles sont étiquetées par des symboles terminaux.
- Les noeuds internes sont étiquetés par des symboles non terminaux

Arbre de dérivation

Pour une grammaire algébrique $G = (\Sigma, V, S, P)$ un arbre de dérivation pour un mot w à partir du symbole S est un arbre enraciné tel que :

- la racine est étiquetée par S
- Les feuilles sont étiquetées par des symboles terminaux.
- Les noeuds internes sont étiquetés par des symboles non terminaux
- Pour chaque noeud interne, si X est son étiquette et $Y_1, Y_2, \ldots, Y_n$ sont les etiquettes de ses fils (dans cet ordre), $X \to Y_1 Y_2 \ldots Y_n$ est une règle de production dans P.

Arbre de dérivation

Pour une grammaire algébrique $G = (\Sigma, V, S, P)$ un arbre de dérivation pour un mot w à partir du symbole S est un arbre enraciné tel que :

- la racine est étiquetée par S
- Les feuilles sont étiquetées par des symboles terminaux.
- Les noeuds internes sont étiquetés par des symboles non terminaux
- Pour chaque noeud interne, si X est son étiquette et $Y_1, Y_2, \ldots, Y_n$ sont les etiquettes de ses fils (dans cet ordre), $X \to Y_1 Y_2 \ldots Y_n$ est une règle de production dans P.
- Le mot obtenu en lisant les feuilles de gauche à droite est le mot w.

Arbre de dérivation

Pour une grammaire algébrique $G = (\Sigma, V, S, P)$ un arbre de dérivation pour un mot w à partir du symbole S est un arbre enraciné tel que :

- la racine est étiquetée par S
- Les feuilles sont étiquetées par des symboles terminaux.
- Les noeuds internes sont étiquetés par des symboles non terminaux
- Pour chaque noeud interne, si X est son étiquette et $Y_1, Y_2, \ldots, Y_n$ sont les etiquettes de ses fils (dans cet ordre), $X \to Y_1 Y_2 \ldots Y_n$ est une règle de production dans P.
- Le mot obtenu en lisant les feuilles de gauche à droite est le mot w.

A toute dérivation correspond un arbre de dérivation.

Arbre de dérivation - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \to aS \mid aSbS \mid \varepsilon$$

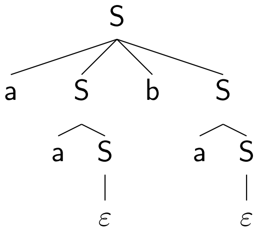
Arbre de dérivation - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \to aS \mid aSbS \mid \varepsilon$$

Voici un exemple de dérivation et d'arbre associé.

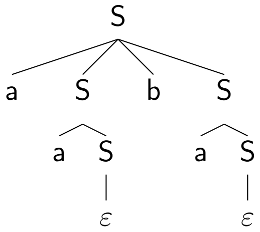
$$\underline{S} \to aSb\underline{S} \to aSba\underline{S} \to a\underline{S}ba \to aa\underline{S}ba \to aaba$$



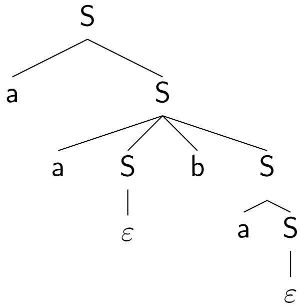
Dérivation la plus à gauche

Attention pour un même mot, il peut y avoir plusieurs dérivations avec des arbres différents

$$\begin{aligned}\underline{S} &\rightarrow aSb\underline{S} \rightarrow aSba\underline{S} \rightarrow a\underline{S}ba \\ &\rightarrow aa\underline{S}ba \rightarrow aaba\end{aligned}$$



$$\begin{aligned}\underline{S} &\rightarrow a\underline{S} \rightarrow aa\underline{S}bS \rightarrow aab\underline{S} \\ &\rightarrow aaba\underline{S} \rightarrow aaba\end{aligned}$$



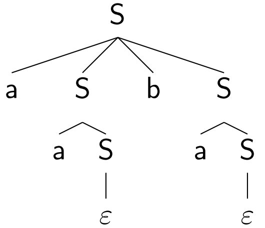
Attention aussi plusieurs dérivations d'un même mot peuvent donner le même arbre

$$\underline{S} \;\to aSb\underline{S} \to aSba\underline{S} \to a\underline{S}ba \to aa\underline{S}ba \to aaba$$

et

$$\underline{S} \;\to a\underline{S}bS \to aa\underline{S}bS \to aab\underline{S} \to aaba\underline{S} \to aaba$$

correspondent toutes les deux à l'arbre :



Pour un arbre donné, une dérivation correspond à un parcours de l'arbre. Pour en choisir une, on peut décider qu'à chaque étape de la dérivation, c'est le symbole non terminal le plus à gauche qui est dérivé. La dérivation obtenue est appelée dérivation la plus à gauche. Elle correspond donc à un parcours préfixe de l'arbre.

Ambiguité

On est surtout intéréssé par l'arbre, donc le fait qu'à un même arbre puisse correspondre plusieurs dérivations, c'est pas génant, on peut toujours considérer la dérivation la plus à gauche.

Ambiguité

On est surtout intéréssé par l'arbre, donc le fait qu'à un même arbre puisse correspondre plusieurs dérivations, c'est pas génant, on peut toujours considérer la dérivation la plus à gauche.

En revanche, le fait que pour un mot donné, lle fait qu'il puisse y avoir plusieurs arbres et génant, par exemple en compilation lorsque l'analyseur syntaxique va essayer de reconstruire l'arbre.

Ambiguité

On est surtout intéréssé par l'arbre, donc le fait qu'à un même arbre puisse correspondre plusieurs dérivations, c'est pas génant, on peut toujours considérer la dérivation la plus à gauche.

En revanche, le fait que pour un mot donné, lle fait qu'il puisse y avoir plusieurs arbres et génant, par exemple en compilation lorsque l'analyseur syntaxique va essayer de reconstruire l'arbre.

Définition

Si tout mot de $L(G)$ possède une unique arbre de dérivation (ou de façon équivalente une unique dérivation la plus à gauche), on dit que la grammaire est non ambiguë.

Un même langage peut avoir des grammaires qui l'engendrent qui sont ambigues et d'autres qui ne le sont pas.

Ambiguité - Exemple

La grammaire G donnée par l'axiome S et les règles

$$S \to aSb \mid aS \mid \varepsilon$$

génère les

Ambiguité - Exemple

La grammaire G donnée par l'axiome S et les règles

$$S \to aSb \mid aS \mid \varepsilon$$

génère les $a^m b^n$ avec $m \geq n$.

Ambiguité - Exemple

La grammaire G donnée par l'axiome S et les règles

$$S \to aSb \mid aS \mid \varepsilon$$

génère les $a^m b^n$ avec $m \geq n$.
C'est une grammaire ambiguë (pourquoi ?)

Ambiguité - Exemple

La grammaire G donnée par l'axiome S et les règles

$$S \to aSb \mid aS \mid \varepsilon$$

génère les $a^m b^n$ avec $m \geq n$.
C'est une grammaire ambiguë (pourquoi ?)

Cependant G' donnée par les règles

$$S \to aSb \mid aT \mid \varepsilon$$
$$T \to aT \mid \varepsilon$$

est non ambiguë et génère le même langage.

Ambiguité

Mentionnons deux résultats sans en donner les preuves :

Lemme (Conséquence du lemme de Parikh)

Il existe des langages algébriques tels que toute grammaire les génère est ambiguë.
Un exemple est le langage $L = \{a^p b^q c^r,\ p = q \text{ ou } q = r\}$.
Un tel langage est dit inhéremment ambigu.

Théoreme

Il n'existe pas d'algorithme pour décider si une grammaire est ambigue ou non.

Algorithmique

La question cruciale une fois une grammaire définie, est de décider si un mot donné appartient ou non au langage décrit par cette grammaire.

La question cruciale une fois une grammaire définie, est de décider si un mot donné appartient ou non au langage décrit par cette grammaire.

Afin de résoudre cette question (et d'autres), il est souvent utile de transformer la grammaire afin de la mettre sous une forme un peu standardisée, on parle de forme normale.

Grammaires réduites

Définition

Un symbole non terminal X d'une grammaire algébrique, G est dit

- *Productif si il existe un mot $u \in \Sigma^*$ tel que $X \overset{*}{\Rightarrow} u$*
- *Accessible si $S \overset{*}{\Rightarrow} uXv$ avec u et v dans V^*.*
- *Utile si il est productif et si $S \overset{*}{\Rightarrow} uXv$ avec u et v ne contenant que des symboles terminaux ou des symboles non teminaux productifs.*

Une grammaire est réduite si tous ses symboles non terminaux sont utiles.

On va voir au transparent suivant un algorithme pour supprimer les symboles non utiles d'une grammaire (et donc toutes les règles de production où ils apparaissent) sans changer le langage généré.

Grammaires réduites

Définition

Un symbole non terminal X d'une grammaire algébrique, G est dit

- *Productif si il existe un mot $u \in \Sigma^*$ tel que $X \overset{*}{\Rightarrow} u$*
- *Accessible si $S \overset{*}{\Rightarrow} uXv$ avec u et v dans V^*.*
- *Utile si il est productif et si $S \overset{*}{\Rightarrow} uXv$ avec u et v ne contenant que des symboles terminaux ou des symboles non teminaux productifs.*

Une grammaire est réduite si tous ses symboles non terminaux sont utiles.

On va voir au transparent suivant un algorithme pour supprimer les symboles non utiles d'une grammaire (et donc toutes les règles de production où ils apparaissent) sans changer le langage généré.

Toute grammaire est donc équivalente à une grammaire réduite.

Définition

Un symbole non terminal X d'une grammaire algébrique, G est dit

- *Productif si il existe un mot $u \in \Sigma^*$ tel que $X \overset{*}{\Rightarrow} u$*
- *Accessible si $S \overset{*}{\Rightarrow} uXv$ avec u et v dans V^*.*
- *Utile si il est productif et si $S \overset{*}{\Rightarrow} uXv$ avec u et v ne contenant que des symboles terminaux ou des symboles non teminaux productifs.*

Une grammaire est réduite si tous ses symboles non terminaux sont utiles.

On va voir au transparent suivant un algorithme pour supprimer les symboles non utiles d'une grammaire (et donc toutes les règles de production où ils apparaissent) sans changer le langage généré.

Toute grammaire est donc équivalente à une grammaire réduite.
Il s'agit de l'équivalent pour les grammaires des automates émondés (tous les états sont accessibles et co-accessibles)

Algorithme de Réduction d'une Grammaire

Pour réduire une grammaire on va appliquer l'algorithme suivant :

1. Déterminer l'ensemble des symboles non terminaux qui sont productifs

Algorithme de Réduction d'une Grammaire

Pour réduire une grammaire on va appliquer l'algorithme suivant :

1. Déterminer l'ensemble des symboles non terminaux qui sont productifs
2. Supprimer les symboles non terminaux non productifs ainsi que toutes les règles où ils apparaissent

Algorithme de Réduction d'une Grammaire

Pour réduire une grammaire on va appliquer l'algorithme suivant :

1. Déterminer l'ensemble des symboles non terminaux qui sont productifs
2. Supprimer les symboles non terminaux non productifs ainsi que toutes les règles où ils apparaissent
3. Si S est improductif, la grammaire ne génère aucun mot on peut la remplacer par une grammaire triviale

Algorithme de Réduction d'une Grammaire

Pour réduire une grammaire on va appliquer l'algorithme suivant :

1. Déterminer l'ensemble des symboles non terminaux qui sont productifs
2. Supprimer les symboles non terminaux non productifs ainsi que toutes les règles où ils apparaissent
3. Si S est improductif, la grammaire ne génère aucun mot on peut la remplacer par une grammaire triviale
4. Si S est productif, déterminer les symboles non terminaux accessibles

Algorithme de Réduction d'une Grammaire

Pour réduire une grammaire on va appliquer l'algorithme suivant :

1. Déterminer l'ensemble des symboles non terminaux qui sont productifs
2. Supprimer les symboles non terminaux non productifs ainsi que toutes les règles où ils apparaissent
3. Si S est improductif, la grammaire ne génère aucun mot on peut la remplacer par une grammaire triviale
4. Si S est productif, déterminer les symboles non terminaux accessibles
5. Supprimer les symboles non terminaux non accessibles ainsi que toutes les règles où ils apparaissent

Algorithme de Réduction d'une Grammaire

Pour réduire une grammaire on va appliquer l'algorithme suivant :

1. Déterminer l'ensemble des symboles non terminaux qui sont productifs
2. Supprimer les symboles non terminaux non productifs ainsi que toutes les règles où ils apparaissent
3. Si S est improductif, la grammaire ne génère aucun mot on peut la remplacer par une grammaire triviale
4. Si S est productif, déterminer les symboles non terminaux accessibles
5. Supprimer les symboles non terminaux non accessibles ainsi que toutes les règles où ils apparaissent

Réduction - Déterminer les symboles productifs

On construit l'ensemble de symboles de proche en proche, tout comme on déterminerait dans un automate les états qui sont co-accessibles.

1. Poser $V_0 = \emptyset$

Réduction - Déterminer les symboles productifs

On construit l'ensemble de symboles de proche en proche, tout comme on déterminerait dans un automate les états qui sont co-accessibles.

1. Poser $V_0 = \emptyset$
2. Calculer successivement les ensembles

$$V_{i+1} = \{X \in V \text{ tels qu'il existe une règle } X \to u, \text{ où } u \in (\Sigma \cup V_i)^*\}$$

Réduction - Déterminer les symboles productifs

On construit l'ensemble de symboles de proche en proche, tout comme on déterminerait dans un automate les états qui sont co-accessibles.

1. Poser $V_0 = \emptyset$
2. Calculer successivement les ensembles

$$V_{i+1} = \{X \in V \text{ tels qu'il existe une règle } X \to u, \text{ où } u \in (\Sigma \cup V_i)^*\}$$

3. Arrêter lorsque $V_{i+1} = V_i$.

Réduction - Déterminer les symboles productifs

On construit l'ensemble de symboles de proche en proche, tout comme on déterminerait dans un automate les états qui sont co-accessibles.

1. Poser $V_0 = \emptyset$
2. Calculer successivement les ensembles

$$V_{i+1} = \{X \in V \text{ tels qu'il existe une règle } X \to u, \text{ où } u \in (\Sigma \cup V_i)^*\}$$

3. Arrêter lorsque $V_{i+1} = V_i$.
4. L'ensemble V_i est alors l'ensemble des symboles productifs.

Réduction - Déterminer les symboles accessibles

Là encore on fait un parcours en largeur, ici à partir de S, pour déterminer les symboles accessibes.

1. Poser $W_0 = \{S\}$

Réduction - Déterminer les symboles accessibles

Là encore on fait un parcours en largeur, ici à partir de S, pour déterminer les symboles accessibes.

1. Poser $W_0 = \{S\}$
2. Calculer successivement les ensembles

$$W_{i+1} = W_i \cup \{X \in V \text{ tels qu'il existe une règle } Y \to uXv, \text{ où } Y \in W_i\}.$$

Là encore on fait un parcours en largeur, ici à partir de S, pour déterminer les symboles accessibes.

1. Poser $W_0 = \{S\}$
2. Calculer successivement les ensembles

$$W_{i+1} = W_i \cup \{X \in V \text{ tels qu'il existe une règle } Y \to uXv\text{, où } Y \in W_i\}.$$

3. Arrêter lorsque $W_{i+1} = W_i$.

Réduction - Déterminer les symboles accessibles

Là encore on fait un parcours en largeur, ici à partir de S, pour déterminer les symboles accessibes.

1. Poser $W_0 = \{S\}$
2. Calculer successivement les ensembles

$$W_{i+1} = W_i \cup \{X \in V \text{ tels qu'il existe une règle } Y \to uXv, \text{ où } Y \in W_i\}.$$

3. Arrêter lorsque $W_{i+1} = W_i$.
4. L'ensemble W_i est alors l'ensemble des symboles accessibles

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \rightarrow a \mid X$$
$$X \rightarrow XY$$
$$Y \rightarrow b$$

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$
\begin{aligned}
S &\to a \mid X \\
X &\to XY \\
Y &\to b
\end{aligned}
$$

1. Symboles productifs : $V_0 = \emptyset$,

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$\begin{aligned} S &\to a \mid X \\ X &\to XY \\ Y &\to b \end{aligned}$$

1. Symboles productifs : $V_0 = \emptyset$, $V_1 = \{S, Y\}$,

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \to a \mid X$$
$$X \to XY$$
$$Y \to b$$

1. Symboles productifs : $V_0 = \emptyset$, $V_1 = \{S, Y\}$, $V_2 = V_1$

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \to a \mid X$$
$$X \to XY$$
$$Y \to b$$

1. Symboles productifs : $V_0 = \emptyset$, $V_1 = \{S, Y\}$, $V_2 = V_1$
2. On supprime donc X pour obtenir la grammaire
   $S \to a$
   $Y \to b$

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$
\begin{aligned}
S &\rightarrow a \mid X \\
X &\rightarrow XY \\
Y &\rightarrow b
\end{aligned}
$$

1. Symboles productifs : $V_0 = \emptyset$, $V_1 = \{S, Y\}$,$V_2 = V_1$
2. On supprime donc X pour obtenir la grammaire
   $S \rightarrow a$
   $Y \rightarrow b$
3. Symboles accessibles : $W_0 = \{S\}$,

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \to a \mid X$$
$$X \to XY$$
$$Y \to b$$

1. Symboles productifs : $V_0 = \emptyset$, $V_1 = \{S, Y\}$,$V_2 = V_1$
2. On supprime donc X pour obtenir la grammaire
   $S \to a$
   $Y \to b$
3. Symboles accessibles : $W_0 = \{S\}$, $W_1 = \{S\}$.

Réduction d'une Grammaire - Exemple

Exemple avec la grammaire sur $\Sigma = \{a, b\}$ donnée par les règles de production suivantes :

$$S \to a \mid X$$
$$X \to XY$$
$$Y \to b$$

1. Symboles productifs : $V_0 = \emptyset$, $V_1 = \{S, Y\}$,$V_2 = V_1$
2. On supprime donc X pour obtenir la grammaire
   $S \to a$
   $Y \to b$
3. Symboles accessibles : $W_0 = \{S\}$, $W_1 = \{S\}$.
4. On supprime donc Y pour obtenir la grammaire
   $S \to a$

Grammaires propres

Définition

Une grammaire algébrique est dite propre si elle ne contient

- *pas de règle unitaire :* $A \to B$ *où* A *et* B *non terminaux.*
- *pas de règle* ε *:* $A \to \varepsilon$

Grammaires propres

Définition

Une grammaire algébrique est dite propre si elle ne contient

- *pas de règle unitaire :* $A \to B$ *où* A *et* B *non terminaux.*
- *pas de règle* ε *:* $A \to \varepsilon$

Théoreme

Tout langage algébrique L ne contenant pas le mot vide peut être engendré par une grammaire propre

(si on s'intéresse à un langage algébrique qui contient le mot vide on peut dire une grammaire propre plus une règle $S \to \varepsilon$ et S n'apparaît pas dans une partie droite d'une règle de production).

Problème du mot

Corollaire (Décidabilité du Problème du mot)

Pour toute grammaire algébrique G et tout mot u, il existe un algorithme qui en temps fini répond oui si u peut etre engendré par G, et non sinon

Corollaire (Décidabilité du Problème du mot)

Pour toute grammaire algébrique G et tout mot u, il existe un algorithme qui en temps fini répond oui si u peut etre engendré par G, et non sinon

Une fois la grammaire propre, on remarque que toute règle est soit du type $X \to a$, soit son membre droit a longueur au moins 2. Cela implique qu'une suite de dérivation qui amène au mot u a longueur au plus $|u| - 1$. Il suffit donc de générer toutes ces dérivations possibles (il y en a bien un nombre fini) pour voir si u peut etre engendré.

Corollaire (Décidabilité du Problème du mot)

Pour toute grammaire algébrique G et tout mot u, il existe un algorithme qui en temps fini répond oui si u peut etre engendré par G, et non sinon

Une fois la grammaire propre, on remarque que toute règle est soit du type $X \rightarrow a$, soit son membre droit a longueur au moins 2. Cela implique qu'une suite de dérivation qui amène au mot u a longueur au plus $|u| - 1$. Il suffit donc de générer toutes ces dérivations possibles (il y en a bien un nombre fini) pour voir si u peut etre engendré.

Remarque 1 : ce nombre de dérivations peut être exponentiel.
Remarque 2 : ceci aurait aussi marché pour les grammaires de type 1

Problème du mot

Corollaire (Décidabilité du Problème du mot)

Pour toute grammaire algébrique G et tout mot u, il existe un algorithme qui en temps fini répond oui si u peut etre engendré par G, et non sinon

Une fois la grammaire propre, on remarque que toute règle est soit du type $X \to a$, soit son membre droit a longueur au moins 2. Cela implique qu'une suite de dérivation qui amène au mot u a longueur au plus $|u| - 1$. Il suffit donc de générer toutes ces dérivations possibles (il y en a bien un nombre fini) pour voir si u peut etre engendré.

Remarque 1 : ce nombre de dérivations peut être exponentiel.
Remarque 2 : ceci aurait aussi marché pour les grammaires de type 1

Mentionnons sans preuve le résultat suivant

Théoreme

Il existe un algorithme polynomial pour résoudre ce problème pour les grammaires algébriques.

Nettoyage d'une Grammaire - Etape 1

1. On calcule les symboles annulables, cad les symboles X tels que $X \Rightarrow \varepsilon$. Comme pour la réduction on pose $E_0 = \{\varepsilon\}$ et on calcule successivement les ensembles $E_{i+1} = \{X \in V$ tels que $(X \to u) \in P$, où $u \in (E_i)^*\}$ jusqu'à ce que $E_{i+1} = E_i$.

1. On calcule les symboles <u>annulables</u>, cad les symboles X tels que $X \Rightarrow \varepsilon$. Comme pour la réduction on pose $E_0 = \{\varepsilon\}$ et on calcule successivement les ensembles $E_{i+1} = \{X \in V \text{ tels que } (X \rightarrow u) \in P, \text{ où } u \in (E_i)^*\}$ jusqu'à ce que $E_{i+1} = E_i$.
2. Pour chaque règle $Y \rightarrow uXv$ où X annulable, on ajoute une règle $Y \rightarrow uv$.

Nettoyage d'une Grammaire - Etape 1

1. On calcule les symboles <u>annulables</u>, cad les symboles X tels que $X \Rightarrow \varepsilon$. Comme pour la réduction on pose $E_0 = \{\varepsilon\}$ et on calcule successivement les ensembles $E_{i+1} = \{X \in V$ tels que $(X \rightarrow u) \in P$, où $u \in (E_i)^*\}$ jusqu'à ce que $E_{i+1} = E_i$.
2. Pour chaque règle $Y \rightarrow uXv$ où X annulable, on ajoute une règle $Y \rightarrow uv$.
3. On supprime toutes les regles $X \rightarrow \varepsilon$

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \overset{*}{\Rightarrow} Y$ et $Y \overset{*}{\Rightarrow} X$.

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \stackrel{*}{\Rightarrow} Y$ et $Y \stackrel{*}{\Rightarrow} X$.
2. Il s'agit d'une relation d'équivalence entre les symboles non terminaux. On choisit donc un représentant pour chaque classe et on le substitue à chaque symbole de sa classe dans toutes les règles de production.

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \overset{*}{\Rightarrow} Y$ et $Y \overset{*}{\Rightarrow} X$.
2. Il s'agit d'une relation d'équivalence entre les symboles non terminaux. On choisit donc un représentant pour chaque classe et on le substitue à chaque symbole de sa classe dans toutes les règles de production.
3. Il en résulte peut-être des règles du type $X \to X$. On les supprime.

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \stackrel{*}{\Rightarrow} Y$ et $Y \stackrel{*}{\Rightarrow} X$.
2. Il s'agit d'une relation d'équivalence entre les symboles non terminaux. On choisit donc un représentant pour chaque classe et on le substitue à chaque symbole de sa classe dans toutes les règles de production.
3. Il en résulte peut-être des règles du type $X \rightarrow X$. On les supprime.
4. Comme on a supprimé les symboles équivalents a l'étape précédente, la relation $X \stackrel{*}{\Rightarrow} Y$ est bien un ordre partiel sur les symboles non terminaux (X est supérieur à Y).

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \stackrel{*}{\Rightarrow} Y$ et $Y \stackrel{*}{\Rightarrow} X$.
2. Il s'agit d'une relation d'équivalence entre les symboles non terminaux. On choisit donc un représentant pour chaque classe et on le substitue à chaque symbole de sa classe dans toutes les règles de production.
3. Il en résulte peut-être des règles du type $X \to X$. On les supprime.
4. Comme on a supprimé les symboles équivalents a l'étape précédente, la relation $X \stackrel{*}{\Rightarrow} Y$ est bien un ordre partiel sur les symboles non terminaux (X est supérieur à Y).
5. On prend le symbole le plus petit Y et on fait les opérations suivantes : pour chaque règle unitaire $X \to Y$ qu'on veut supprimer et pour chaque règle $Y \to u$, on ajoute une règle $X \to u$.

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \stackrel{*}{\Rightarrow} Y$ et $Y \stackrel{*}{\Rightarrow} X$.
2. Il s'agit d'une relation d'équivalence entre les symboles non terminaux. On choisit donc un représentant pour chaque classe et on le substitue à chaque symbole de sa classe dans toutes les règles de production.
3. Il en résulte peut-être des règles du type $X \to X$. On les supprime.
4. Comme on a supprimé les symboles équivalents a l'étape précédente, la relation $X \stackrel{*}{\Rightarrow} Y$ est bien un ordre partiel sur les symboles non terminaux (X est supérieur à Y).
5. On prend le symbole le plus petit Y et on fait les opérations suivantes : pour chaque règle unitaire $X \to Y$ qu'on veut supprimer et pour chaque règle $Y \to u$, on ajoute une règle $X \to u$. Ensuite on peut supprimer la règle unitaire $X \to Y$.

Nettoyage d'une Grammaire - Etape 2

Ensuite on supprime toutes les règles unitaires.

1. On identifie toutes les paires de symboles non terminaux X et Y tels que $X \stackrel{*}{\Rightarrow} Y$ et $Y \stackrel{*}{\Rightarrow} X$.
2. Il s'agit d'une relation d'équivalence entre les symboles non terminaux. On choisit donc un représentant pour chaque classe et on le substitue à chaque symbole de sa classe dans toutes les règles de production.
3. Il en résulte peut-être des règles du type $X \rightarrow X$. On les supprime.
4. Comme on a supprimé les symboles équivalents a l'étape précédente, la relation $X \stackrel{*}{\Rightarrow} Y$ est bien un ordre partiel sur les symboles non terminaux (X est supérieur à Y).
5. On prend le symbole le plus petit Y et on fait les opérations suivantes : pour chaque règle unitaire $X \rightarrow Y$ qu'on veut supprimer et pour chaque règle $Y \rightarrow u$, on ajoute une règle $X \rightarrow u$. Ensuite on peut supprimer la règle unitaire $X \rightarrow Y$.
6. On recommence avec le nouveau symbole minimal jusqu'à ce qu'on ait tout nettoyé.

Nettoyage d'une Grammaire - Exemple

Prenons l'exemple de la grammaire $(\Sigma = \{a, b\}, V = \{S, A, B\}, S, P)$ ou les règles P sont

$$
\begin{aligned}
S &\rightarrow aAB \mid BA \mid b \\
A &\rightarrow BBB \mid a \\
B &\rightarrow AB \mid b \mid \varepsilon
\end{aligned}
$$

Nettoyage d'une Grammaire - Exemple

Prenons l'exemple de la grammaire $(\Sigma = \{a, b\}, V = \{S, A, B\}, S, P)$ ou les règles P sont

$$\begin{aligned} S &\rightarrow aAB \mid BA \mid b \\ A &\rightarrow BBB \mid a \\ B &\rightarrow AB \mid b \mid \varepsilon \end{aligned}$$

Suite au tableau ....

Forme normale de Chomsky

Une grammaire algébrique est sous Forme Normale de Chomsky (FNC) si toute production est de la forme

$$\begin{array}{ll} A \to BC & A, B, C \in V \\ A \to a & a \in \Sigma \end{array}$$

L'arbre de dérivation est donc un arbre binaire.
L'intérêt est théorique, cela permet de simplifier certaines preuves en évitant de devoir faire de nombreux cas.

Forme normale de Chomsky - Algorithme

On commence par transformer la grammaire sous forme réduite et propre.

Forme normale de Chomsky - Algorithme

On commence par transformer la grammaire sous forme réduite et propre.

Pour chaque lettre a dans Σ (i.e. chaque symbole terminal), on ajoute un symbole non productif X_a avec la règle $X_a \to a$

Forme normale de Chomsky - Algorithme

On commence par transformer la grammaire sous forme réduite et propre.

Pour chaque lettre a dans Σ (i.e. chaque symbole terminal), on ajoute un symbole non productif X_a avec la règle $X_a \to a$

Dans toutes les anciennes règles, on remplace a par X_a

Forme normale de Chomsky - Algorithme

On commence par transformer la grammaire sous forme réduite et propre.

Pour chaque lettre a dans Σ (i.e. chaque symbole terminal), on ajoute un symbole non productif X_a avec la règle $X_a \to a$

Dans toutes les anciennes règles, on remplace a par X_a

Toutes les règles sont donc maintenant de la forme $X \to a$ ou de la forme $X \to X_1 X_2 ... X_n$.

Forme normale de Chomsky - Algorithme

On commence par transformer la grammaire sous forme réduite et propre.

Pour chaque lettre a dans Σ (i.e. chaque symbole terminal), on ajoute un symbole non productif X_a avec la règle $X_a \to a$

Dans toutes les anciennes règles, on remplace a par X_a

Toutes les règles sont donc maintenant de la forme $X \to a$ ou de la forme $X \to X_1 X_2 ... X_n$.

Ensuite il suffit de remplacer toute règle $X \to X_1 X_2 ... X_n$ avec $n > 2$ par

$$
\begin{aligned}
&X \to X_1 Y_1 \\
&Y_1 \to X_2 Y_2 \\
&Y_2 \to X_3 Y_3 \\
&\ldots \\
&Y_{n-2} \to X_{n-1} X_n
\end{aligned}
$$

Lemme d'Itération

Commençons par les deux observations suivantes (faire un dessin ! ! !)

Lemme

Soit G est une grammaire algébrique ayant p symboles non terminaux.
Si la dérivation d'un mot w a profondeur supérieure ou égale à $p+1$ alors il existe un symbole non terminal X apparaissant deux fois sur une meme branche dans la dérivation de w.

Lemme d'Itération

Commençons par les deux observations suivantes (faire un dessin ! ! !)

Lemme

*Soit G est une grammaire algébrique ayant p symboles non terminaux.
Si la dérivation d'un mot w a profondeur supérieure ou égale à $p+1$ alors il existe un symbole non terminal X apparaissant deux fois sur une meme branche dans la dérivation de w.*

Lemme

*Soit G est une grammaire sous forme normale de Chomsky, et soit w un mot généré par cette grammaire.
Si $|w| > 2^{p-1}$ alors tout arbre de dérivation pour ce mot dans cette grammaire a profondeur au moins $p+1$.*

Lemme d'Itération

Commençons par les deux observations suivantes (faire un dessin ! ! !)

Lemme

Soit G est une grammaire algébrique ayant p symboles non terminaux.
Si la dérivation d'un mot w a profondeur supérieure ou égale à $p+1$ alors il existe un symbole non terminal X apparaissant deux fois sur une meme branche dans la dérivation de w.

Lemme

Soit G est une grammaire sous forme normale de Chomsky, et soit w un mot généré par cette grammaire.
Si $|w| > 2^{p-1}$ alors tout arbre de dérivation pour ce mot dans cette grammaire a profondeur au moins $p+1$.

En combinant les lemmes précédent on peut démontrer :

Théoreme (Lemme d'itération pour les langages algébriques)

Si L est algébrique, alors il existe un N tel que pout tout mot $w \in L$ de longueur au moins N, il existe une décomposition $w = xuyvz$ avec

- $uv \neq \varepsilon$
- $|uyv| < N$
- $\forall n \in \mathbb{N},\ xu^nyv^nz \in L$

On prend $N = 2^{p-1} + 1$ ou p est le nombre de symboles non terminaux dans une grammaire sous forme de Chomsky qui engendre L.

Lemme d'Itération

En combinant les lemmes précédent on peut démontrer :

Théoreme (Lemme d'itération pour les langages algébriques)

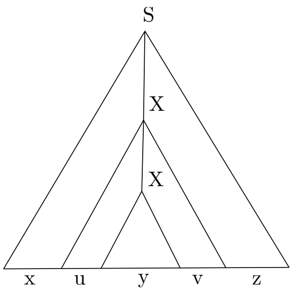
Si L est algébrique, alors il existe un N tel que pout tout mot $w \in L$ de longueur au moins N, il existe une décomposition $w = xuyvz$ avec

- $uv \neq \varepsilon$
- $|uyv| < N$
- $\forall n \in I\!N,\ xu^nyv^nz \in L$

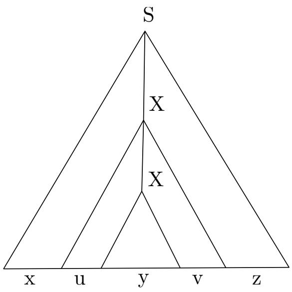
On prend $N = 2^{p-1} + 1$ ou p est le nombre de symboles non terminaux dans une grammaire sous forme de Chomsky qui engendre L.

On trouve donc dans la dérivation de w deux symboles non productifs identiques dans une meme branche partant de la racine.

Lemme d'Itération

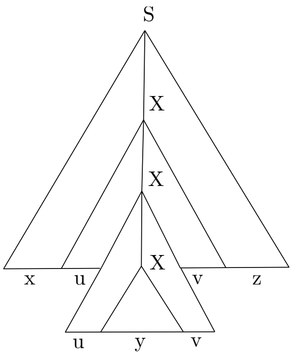


Lemme d'Itération



Le symbole X apparait au moins deux fois sur la meme branche.
On considère le sous arbre enraciné en le X le plus bas sur la branche, et on lui subsititue le sous arbre enraciné en X plus haut sur la branche. On obtient bien une dérivation du mot xuuyvvz. On peut réitérer l'opération.

Lemme d'Itération



Le symbole X apparait au moins deux fois sur la meme branche.
On considère le sous arbre enraciné en le X le plus bas sur la branche, et on lui subsititue le sous arbre enraciné en X plus haut sur la branche. On obtient bien une dérivation du mot xuuyvvz. On peut réitérer l'opération.

Corollaire

le langage $\{a^n b^n c^n,\ n \in \mathbb{N}\}$ *n'est pas algébrique.*

Décidabilité du Problème du vide

Théoreme

Il existe un algorihtme permettant de décider si une grammaire algébrique engendre au moins un mot.

Décidabilité du Problème du vide

Théoreme

Il existe un algorihtme permettant de décider si une grammaire algébrique engendre au moins un mot.

La preuve est similaire à celle du lemme d'itération : si un mot u admet un arbre de dérivation, alors soit aucun symbole non productif n'apparait plus d'une fois par branche, soit il existe un mot plus court qui admet aussi un arbre de dérivation.

Décidabilité du Problème du vide

Théoreme

Il existe un algorihtme permettant de décider si une grammaire algébrique engendre au moins un mot.

La preuve est similaire à celle du lemme d'itération : si un mot u admet un arbre de dérivation, alors soit aucun symbole non productif n'apparait plus d'une fois par branche, soit il existe un mot plus court qui admet aussi un arbre de dérivation.

Ainsi si le langage engendré est non vide, il existe un mot engendré tel qu'aucun symbole non productif n'apparait plus d'une fois par branche.

Décidabilité du Problème du vide

Théoreme

Il existe un algorihtme permettant de décider si une grammaire algébrique engendre au moins un mot.

La preuve est similaire à celle du lemme d'itération : si un mot u admet un arbre de dérivation, alors soit aucun symbole non productif n'apparait plus d'une fois par branche, soit il existe un mot plus court qui admet aussi un arbre de dérivation.

Ainsi si le langage engendré est non vide, il existe un mot engendré tel qu'aucun symbole non productif n'apparait plus d'une fois par branche.

Il y a un nombre fini de tels arbres, on peut tous les énumérer et voir si un produit un mot.