Théorie des Langages et Compilation : Analyse Syntaxique

Didier Lime

École Centrale de Nantes – LS2N

Année 2017 – 2018

Plan

Plan

Introduction

Grammaires hors contexte

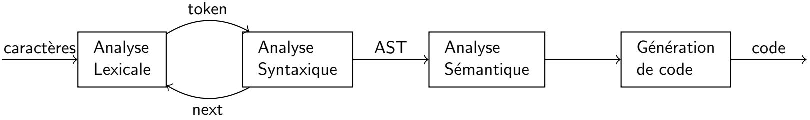
Analyse descendante

Analyse ascendante

Yacc

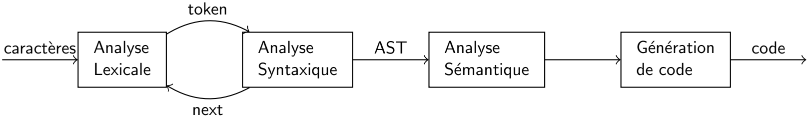
Conclusion

Rôle de l'analyseur syntaxique



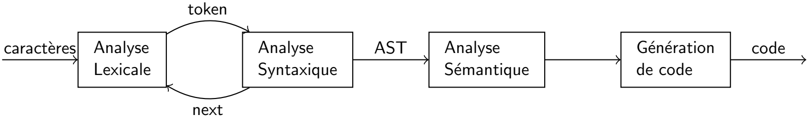
- L'analyseur syntaxique récupère les mots (*tokens*) isolés par l'analyseur lexical ;

Rôle de l'analyseur syntaxique



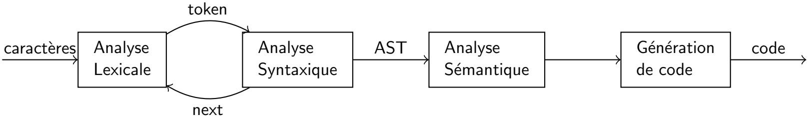
- L'analyseur syntaxique récupère les mots (*tokens*) isolés par l'analyseur lexical ;
- Il vérifie leur **bon agencement** ;

Rôle de l'analyseur syntaxique



- L'analyseur syntaxique récupère les mots (*tokens*) isolés par l'analyseur lexical ;
- Il vérifie leur **bon agencement** ;
- Il produit une **représentation abstraite** de l'entrée pour les phases suivantes (*Abstract Syntax Tree* (AST))

Rôle de l'analyseur syntaxique



- L'analyseur syntaxique récupère les mots (*tokens*) isolés par l'analyseur lexical ;
- Il vérifie leur **bon agencement** ;
- Il produit une **représentation abstraite** de l'entrée pour les phases suivantes (*Abstract Syntax Tree* (AST))

En pratique, on réalise certaines phases de l'analyse sémantique (voire de la génération de code) en même temps que l'analyse syntaxique.

Insuffisance des langages réguliers

- Définition (partielle) de la **syntaxe** d'un langage de programmation :

$$instr ::= \begin{cases} \texttt{if}\ \ expr\ \texttt{then}\ \ instr\ \texttt{endif} \\ \texttt{print}\ \ chaine \end{cases}$$

Insuffisance des langages réguliers

- Définition (partielle) de la **syntaxe** d'un langage de programmation :

$$instr ::= \begin{cases} \texttt{if}\ \ expr\ \texttt{then}\ \ instr\ \texttt{endif} \\ \texttt{print}\ \ chaine \end{cases}$$

- Alphabet associé :

$$\Sigma = \{i, t, e, E, p, c\}$$

Insuffisance des langages réguliers

- Définition (partielle) de la **syntaxe** d'un langage de programmation :

$$instr ::= \begin{cases} \texttt{if} \ \ expr \ \ \texttt{then} \ \ instr \ \ \texttt{endif} \\ \texttt{print} \ \ chaine \end{cases}$$

- Alphabet associé :

$$\Sigma = \{i, t, e, E, p, c\}$$

- Langage associé :

$$L = \{pc, iEtpce, iEtiEtpcee, \dots, (iEt)^k pce^k, \dots\}$$

Insuffisance des langages réguliers

- Définition (partielle) de la **syntaxe** d'un langage de programmation :

$$instr ::= \begin{cases} \texttt{if}\ \ expr\ \ \texttt{then}\ \ instr\ \ \texttt{endif} \\ \texttt{print}\ \ chaine \end{cases}$$

- Alphabet associé :

$$\Sigma = \{i, t, e, E, p, c\}$$

- Langage associé :

$$L = \{pc, iEtpce, iEtiEtpcee, \ldots, (iEt)^k pce^k, \ldots\}$$

- L n'est **pas régulier**

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;
- On définit des **grammaires formelles** :

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;
- On définit des **grammaires formelles** :

Définition

Une grammaire formelle est un quadruplet (N, Σ, P, S) *où :*

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;
- On définit des **grammaires formelles** :

Définition

Une grammaire formelle est un quadruplet (N, Σ, P, S) *où :*

- N *est un ensemble fini de (symboles)* **non terminaux** *;*

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;
- On définit des **grammaires formelles** :

Définition

Une grammaire formelle est un quadruplet (N, Σ, P, S) *où :*

- N *est un ensemble fini de (symboles)* **non terminaux** *;*
- Σ *est un ensemble fini de (symboles)* **terminaux** *;*

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;
- On définit des **grammaires formelles** :

Définition

Une grammaire formelle est un quadruplet (N, Σ, P, S) où :

- *N est un ensemble fini de (symboles)* **non terminaux** *;*
- *Σ est un ensemble fini de (symboles)* **terminaux** *;*
- *P est un ensemble fini de* **règles de production** *de la forme :*

$$(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$$

Grammaires formelles

- On a besoin d'un formalisme plus puissant que les **expressions régulières** ;
- On définit des **grammaires formelles** :

Définition

Une grammaire formelle est un quadruplet (N, Σ, P, S) où :

- *N est un ensemble fini de (symboles)* **non terminaux** *;*
- *Σ est un ensemble fini de (symboles)* **terminaux** *;*
- *P est un ensemble fini de* **règles de production** *de la forme :*

$$(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$$

- *S est un élément de N appelé* **axiome***.*

Grammaires formelles

Exemple

$S \rightarrow AbB$
$A \rightarrow aA$
$A \rightarrow b$
$aAb \rightarrow B$
$BB \rightarrow A$
$B \rightarrow \epsilon$

Dérivations et langage

Soient $x, y \in (\Sigma \cup N)$ et $G = (N, \Sigma, P, S)$ une grammaire.

- y **dérive** de x (en un pas), noté $x \Rightarrow y$ si :

$$\exists u, v, p, q \in (\Sigma \cup N)^* \text{ t.q. } x = upv, y = uqv \text{ et } p \to q \in P$$

Dérivations et langage

Soient $x, y \in (\Sigma \cup N)$ et $G = (N, \Sigma, P, S)$ une grammaire.

- y **dérive** de x (en un pas), noté $x \Rightarrow y$ si :

$$\exists u, v, p, q \in (\Sigma \cup N)^* \text{ t.q. } x = upv, y = uqv \text{ et } p \rightarrow q \in P$$

- On peut généraliser la relation en en prenant la **fermeture réflexive transitive** $\Rightarrow^*$.

Dérivations et langage

Soient $x, y \in (\Sigma \cup N)$ et $G = (N, \Sigma, P, S)$ une grammaire.

- y **dérive** de x (en un pas), noté $x \Rightarrow y$ si :

$$\exists u, v, p, q \in (\Sigma \cup N)^* \text{ t.q. } x = upv, y = uqv \text{ et } p \rightarrow q \in P$$

- On peut généraliser la relation en en prenant la **fermeture réflexive transitive** $\Rightarrow^*$.
- Une **proto-phrase** (*sentential form*) de G est un mot W de $(N \cup \Sigma)^*$ tel que $S \Rightarrow^* W$;

Dérivations et langage

Soient $x, y \in (\Sigma \cup N)$ et $G = (N, \Sigma, P, S)$ une grammaire.

- y **dérive** de x (en un pas), noté $x \Rightarrow y$ si :

$$\exists u, v, p, q \in (\Sigma \cup N)^* \text{ t.q. } x = upv, y = uqv \text{ et } p \rightarrow q \in P$$

- On peut généraliser la relation en en prenant la **fermeture réflexive transitive** $\Rightarrow^*$.
- Une **proto-phrase** (*sentential form*) de G est un mot W de $(N \cup \Sigma)^*$ tel que $S \Rightarrow^* W$;
- Une **phrase** (*sentence*) de G est une proto-phrase ne contenant que des **terminaux** ;

Dérivations et langage

Soient $x, y \in (\Sigma \cup N)$ et $G = (N, \Sigma, P, S)$ une grammaire.

- y **dérive** de x (en un pas), noté $x \Rightarrow y$ si :

$$\exists u, v, p, q \in (\Sigma \cup N)^* \text{ t.q. } x = upv, y = uqv \text{ et } p \rightarrow q \in P$$

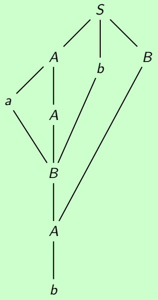
- On peut généraliser la relation en en prenant la **fermeture réflexive transitive** $\Rightarrow^*$.
- Une **proto-phrase** (*sentential form*) de G est un mot W de $(N \cup \Sigma)^*$ tel que $S \Rightarrow^* W$;
- Une **phrase** (*sentence*) de G est une proto-phrase ne contenant que des **terminaux** ;
- Le **langage** de G est l'ensemble de ses phrases.

Dérivations et langage

Exemple

$S \rightarrow AbB$
$A \rightarrow aA$
$A \rightarrow b$
$aAb \rightarrow B$
$BB \rightarrow A$
$B \rightarrow \epsilon$

Graphe de dérivation de $b \in L$:



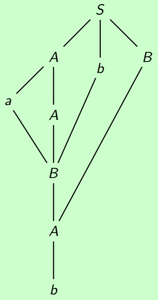
$L =$

Dérivations et langage

Exemple

$S \rightarrow AbB$
$A \rightarrow aA$
$A \rightarrow b$
$aAb \rightarrow B$
$BB \rightarrow A$
$B \rightarrow \epsilon$

Graphe de dérivation de $b \in L$:



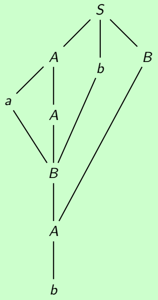
$L = a^*(\epsilon|b|bb)$ *régulier* !

Dérivations et langage

Exemple

$S \rightarrow AbB$
$A \rightarrow aA$
$A \rightarrow b$
$aAb \rightarrow B$
$BB \rightarrow A$
$B \rightarrow \epsilon$

Graphe de dérivation de $b \in L$:



$L = a^*(\epsilon|b|bb)$ *régulier* !

Exemple

$S \rightarrow aSb|c$

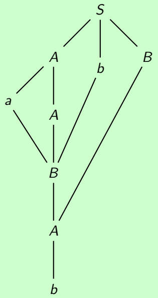
$L =$

Dérivations et langage

Exemple

$S \to AbB$
$A \to aA$
$A \to b$
$aAb \to B$
$BB \to A$
$B \to \epsilon$

Graphe de dérivation de $b \in L$:



$L = a^*(\epsilon|b|bb)$ *régulier* !

Exemple

$S \to aSb|c$

$L = \{a^n c b^n | n \in \mathbb{N}\}$

Dérivations : Exercices

Exercice

Donnez l'arbre de dérivation du mot $w = abaacbbab$ dans la grammaire suivante :

$$\begin{aligned} S &\rightarrow ACB \\ A &\rightarrow aA|b \\ B &\rightarrow Bb|a \\ C &\rightarrow aBcAb \end{aligned}$$

Dérivations : Exercices

Exercice

Donnez l'arbre de dérivation du mot $w = abaacbbab$ dans la grammaire suivante :

$$S \rightarrow ACB$$
$$A \rightarrow aA|b$$
$$B \rightarrow Bb|a$$
$$C \rightarrow aBcAb$$

Exercice

Donnez l'arbre de dérivation du mot $w = acbbcacbb$ dans la grammaire suivante :

$$S \rightarrow ACcaB$$
$$A \rightarrow aB|\epsilon$$
$$B \rightarrow Bb|c$$
$$C \rightarrow bAb|a$$

Hiérarchie de Chomsky

Type 3	Grammaires régulières	$A \rightarrow aB \mid a$ $A \rightarrow Ba \mid a$

$a \in \Sigma,\ A, B \in N$

Hiérarchie de Chomsky

Type 2	Grammaires hors contexte	$A \to W$
Type 3	Grammaires régulières	$A \to aB\|a$
		$A \to Ba\|a$

$a \in \Sigma$, $A, B \in N$ et $W \in (\Sigma \cup N)^*$

Hiérarchie de Chomsky

Type 1	Grammaires contextuelles	$U\mathbf{A}V \rightarrow U\mathbf{W}V$
Type 2	Grammaires hors contexte	$A \rightarrow W$
Type 3	Grammaires régulières	$A \rightarrow aB \mid a$ $A \rightarrow Ba \mid a$

$a \in \Sigma$, $A, B \in N$ et $U, V, W \in (\Sigma \cup N)^*$

Hiérarchie de Chomsky

Type 0	Grammaires générales	$UAV \rightarrow W$
Type 1	Grammaires contextuelles	$U\mathbf{A}V \rightarrow U\mathbf{W}V$
Type 2	Grammaires hors contexte	$A \rightarrow W$
Type 3	Grammaires régulières	$A \rightarrow aB \mid a$
		$A \rightarrow Ba \mid a$

$a \in \Sigma$, $A, B \in N$ et $U, V, W \in (\Sigma \cup N)^*$

Hiérarchie de Chomsky

Type 0	Grammaires générales	$UAV \rightarrow W$
Type 1	Grammaires contextuelles	$U\mathbf{A}V \rightarrow U\mathbf{W}V$
Type 2	**Grammaires hors contexte**	$A \rightarrow W$
Type 3	Grammaires régulières	$A \rightarrow aB\|a$ $A \rightarrow Ba\|a$

$a \in \Sigma$, $A, B \in N$ et $U, V, W \in (\Sigma \cup N)^*$

Plan

Grammaires hors contexte

- Les grammaires **hors contexte** (non contextuelles) offrent :

Grammaires hors contexte

- Les grammaires **hors contexte** (non contextuelles) offrent :
  - un bon **compromis** efficacité / pouvoir d'expression ;

Grammaires hors contexte

- Les grammaires **hors contexte** (non contextuelles) offrent :
  - un bon **compromis** efficacité / pouvoir d'expression ;
  - une bonne **lisibilité**.

Grammaires hors contexte

- Les grammaires **hors contexte** (non contextuelles) offrent :
  - un bon **compromis** efficacité / pouvoir d'expression ;
  - une bonne **lisibilité**.
- Elles sont de la forme :

$$A \to W, W \in (\Sigma \cup N)^*$$

Grammaires hors contexte

- Les grammaires **hors contexte** (non contextuelles) offrent :
  - un bon **compromis** efficacité / pouvoir d'expression ;
  - une bonne **lisibilité**.
- Elles sont de la forme :

$$A \rightarrow W, W \in (\Sigma \cup N)^*$$

Exemple

$$S \rightarrow SAS|(S)|a$$
$$A \rightarrow +|-|*|/$$

$$\Sigma = \{a, +, -, *, /, (,)\}, N = \{S, A\}$$

Ce qu'on peut faire : les grammaires régulières

Théorème

Tout langage **régulier** *est le langage d'une grammaire hors contexte*

Ce qu'on peut faire : les grammaires régulières

Théorème

Tout langage **régulier** *est le langage d'une grammaire hors contexte*

- Un non terminal par **état** de l'automate fini ;

Ce qu'on peut faire : les grammaires régulières

Théorème

Tout langage **régulier** *est le langage d'une grammaire hors contexte*

- Un non terminal par **état** de l'automate fini ;
- Une production $A \rightarrow aB$ par **transition** étiquetée a entre A et B ;

Ce qu'on peut faire : les grammaires régulières

Théorème

Tout langage **régulier** *est le langage d'une grammaire hors contexte*

- Un non terminal par **état** de l'automate fini ;
- Une production $A \rightarrow aB$ par **transition** étiquetée a entre A et B ;
- Une production $F \rightarrow \epsilon$ pour tout état **accepteur** F ;

Ce qu'on peut faire : les grammaires régulières

Théorème

Tout langage **régulier** *est le langage d'une grammaire hors contexte*

- Un non terminal par **état** de l'automate fini ;
- Une production $A \rightarrow aB$ par **transition** étiquetée a entre A et B ;
- Une production $F \rightarrow \epsilon$ pour tout état **accepteur** F ;
- L'axiome est l'état **initial**.

Ce qu'on peut faire : les grammaires régulières

Théorème

Tout langage **régulier** *est le langage d'une grammaire hors contexte*

- Un non terminal par **état** de l'automate fini ;
- Une production $A \rightarrow aB$ par **transition** étiquetée a entre A et B ;
- Une production $F \rightarrow \epsilon$ pour tout état **accepteur** F ;
- L'axiome est l'état **initial**.

Exercice

Écrire une grammaire hors contexte reconnaissant a^*b^*.

Ce qu'on peut faire : des grammaires non régulières

Exemple

$S \rightarrow aSb|\epsilon$

Ce qu'on peut faire : des grammaires non régulières

Exemple

$S \rightarrow aSb|\epsilon$

Exemple

$S \rightarrow (S)S|\epsilon$

Ce qu'on peut faire : des grammaires non régulières

Exemple

$S \to aSb|\epsilon$

Exemple

$S \to (S)S|\epsilon$

Exemple

$S \to iE : Se|wE : Se|x := n|px$
$E \to xAn|EoE|EaE|cE$
$A \to = | < | >$

Ce qu'on peut faire : des grammaires non régulières

Exemple

$S \rightarrow aSb|\epsilon$

Exemple

$S \rightarrow (S)S|\epsilon$

Exemple

$S \rightarrow iE : Se|wE : Se|x := n|px$
$E \rightarrow xAn|EoE|EaE|cE$
$A \rightarrow = | < | >$

Exercice

Écrire une grammaire hors contexte qui engendre $L = \{wcw^R | w \in \{a, b\}^*\}$ avec w^R l'inverse de w (si $w = abb$, $w^R = bba\ldots$)

Ce qu'on ne peut PAS faire : des grammaires contextuelles

Exemple

$\{wcw | w \in \{a, b\}^*\}$ Prédéclaration des variables

Ce qu'on ne peut PAS faire : des grammaires contextuelles

Exemple

$\{wcw \| w \in \{a, b\}^*\}$	Prédéclaration des variables

Exemple

$\{a^n b^m c^n d^m \| m, n \in \mathbb{N} \setminus \{0\}\}$	Vérification nombre d'arguments de deux fonctions (déclarations puis utilisations)

Ce qu'on ne peut PAS faire : des grammaires contextuelles

Exemple

$\{wcw | w \in \{a, b\}^*\}$ Prédéclaration des variables

Exemple

$\{a^n b^m c^n d^m | m, n \in \mathbb{N} \setminus \{0\}\}$ Vérification nombre d'arguments de deux fonctions (déclarations puis utilisations)

Exemple

$\{a^n b^n c^n | n \in \mathbb{N}\}$ Comparaison de la longueur de trois chaînes

Propriétés de fermeture

Si L et L' sont deux langages **hors contexte** :

- Leur **union** $L \cup L'$ est un langage hors contexte ;
- Leur **concaténation** $L.L'$ est un langage hors contexte ;
- Leurs **inverses** sont des langages hors contexte ;

Propriétés de fermeture

Si L et L' sont deux langages **hors contexte** :

- Leur **union** $L \cup L'$ est un langage hors contexte ;
- Leur **concaténation** $L.L'$ est un langage hors contexte ;
- Leurs **inverses** sont des langages hors contexte ;

Mais, en général :

- Leur **intersection** $L \cap L'$ n'est **pas** un langage hors contexte
  Mais si L' est **régulier**, $L \cap L'$ est hors contexte ;
- Leurs **complémentaires** $\overline{L}$ et $\overline{L'}$ ne sont **pas** des langages hors contexte.

Arbres syntaxiques

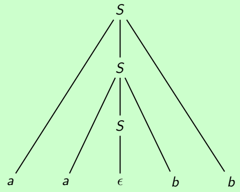
Pour une grammaire hors contexte, le **graphe de dérivation** d'une chaîne de terminaux est un **arbre**

arbre de dérivation = arbre d'analyse = arbre syntaxique

Exemple

$S \rightarrow aSb|\epsilon$

Arbre de dérivation de $aabb$:



Dérivations gauche et droite

- En remplaçant toujours le non-terminal **le plus à gauche**, on obtient une **dérivation gauche** notée $\Rightarrow_g^*$:

Exemple

$S \rightarrow ABC$
$A \rightarrow Aa|a$
$B \rightarrow a|b$
$C \rightarrow c$

$$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$$

Dérivations gauche et droite

- En remplaçant toujours le non-terminal **le plus à gauche**, on obtient une **dérivation gauche** notée $\Rightarrow_g^*$:

Exemple

$S \rightarrow ABC$
$A \rightarrow Aa|a$
$B \rightarrow a|b$
$C \rightarrow c$

$$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$$

- Idem **à droite** ($\Rightarrow_d^*$) :

Exemple

$$S \Rightarrow_d ABC \Rightarrow_d ABc \Rightarrow_d Abc \Rightarrow_d Aabc \Rightarrow_d aabc$$

Dérivations gauche et droite

- En remplaçant toujours le non-terminal **le plus à gauche**, on obtient une **dérivation gauche** notée $\Rightarrow_g^*$:

Exemple

$$S \to ABC$$
$$A \to Aa|a$$
$$B \to a|b$$
$$C \to c$$

$$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$$

- Idem **à droite** ($\Rightarrow_d^*$) :

Exemple

$$S \Rightarrow_d ABC \Rightarrow_d ABc \Rightarrow_d Abc \Rightarrow_d Aabc \Rightarrow_d aabc$$

- Pour une grammaire **hors contexte**, l'ensemble des **phrases** obtenues uniquement par dérivation gauche (ou uniquement droite) est **exactement** son langage.

Grammaire ambiguë

- L'**arbre de dérivation** d'une chaîne ne dépend **pas** de l'**ordre** des dérivations ;

Grammaire ambiguë

- L'**arbre de dérivation** d'une chaîne ne dépend **pas** de l'**ordre** des dérivations ;

Exercice

Construire l'arbre de dérivation commun à :

$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$

$S \Rightarrow_d ABC \Rightarrow_d ABc \Rightarrow_d Abc \Rightarrow_d Aabc \Rightarrow_d aabc$

Grammaire ambiguë

- L'**arbre de dérivation** d'une chaîne ne dépend **pas** de l'**ordre** des dérivations ;

Exercice

Construire l'arbre de dérivation commun à :
$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$
$S \Rightarrow_d ABC \Rightarrow_d ABc \Rightarrow_d Abc \Rightarrow_d Aabc \Rightarrow_d aabc$

- On peut donc ne considérer **que** des dérivations **gauches** (ou que des droites) ;

Grammaire ambiguë

- L'**arbre de dérivation** d'une chaîne ne dépend **pas** de l'**ordre** des dérivations ;

Exercice

Construire l'arbre de dérivation commun à :
$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$
$S \Rightarrow_d ABC \Rightarrow_d ABc \Rightarrow_d Abc \Rightarrow_d Aabc \Rightarrow_d aabc$

- On peut donc ne considérer **que** des dérivations **gauches** (ou que des droites) ;
- Pour tout arbre de dérivation, il existe une dérivation gauche **unique** (idem à droite) ;

Grammaire ambiguë

- L'**arbre de dérivation** d'une chaîne ne dépend **pas** de l'**ordre** des dérivations ;

Exercice

Construire l'arbre de dérivation commun à :

$S \Rightarrow_g ABC \Rightarrow_g AaBC \Rightarrow_g aaBC \Rightarrow_g aabC \Rightarrow_g aabc$

$S \Rightarrow_d ABC \Rightarrow_d ABc \Rightarrow_d Abc \Rightarrow_d Aabc \Rightarrow_d aabc$

- On peut donc ne considérer **que** des dérivations **gauches** (ou que des droites) ;
- Pour tout arbre de dérivation, il existe une dérivation gauche **unique** (idem à droite) ;
- **Mais** pour une phrase donnée, il peut y avoir **plusieurs** arbres de dérivation correspondant
  Et donc plusieurs dérivations gauche ou droite différentes.

Grammaire ambiguë

Exemple

Dérivations de $n + n * n$:

$S \rightarrow S + S$
$S \rightarrow S * S$
$S \rightarrow n$

Grammaire ambiguë

Exemple

Dérivations de $n + n * n$:

$S \Rightarrow_g S + S \Rightarrow_g n + S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \rightarrow S + S$
$S \rightarrow S * S$
$S \rightarrow n$

Grammaire ambiguë

Exemple

Dérivations de $n + n * n$:

$S \Rightarrow_g S + S \Rightarrow_g n + S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \Rightarrow_g S * S \Rightarrow_g S + S * S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \rightarrow S + S$
$S \rightarrow S * S$
$S \rightarrow n$

Grammaire ambiguë

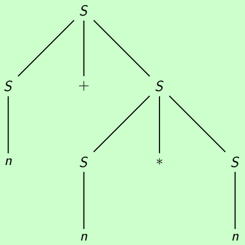
Exemple

Dérivations de $n + n * n$:

$S \Rightarrow_g S + S \Rightarrow_g n + S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \Rightarrow_g S * S \Rightarrow_g S + S * S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \rightarrow S + S$
$S \rightarrow S * S$
$S \rightarrow n$



Grammaire ambiguë

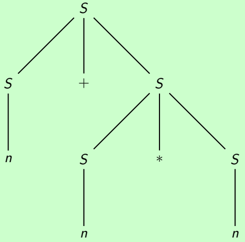
Exemple

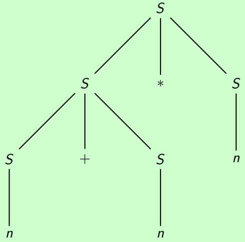
Dérivations de $n + n * n$:

$S \Rightarrow_g S + S \Rightarrow_g n + S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \Rightarrow_g S * S \Rightarrow_g S + S * S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \rightarrow S + S$
$S \rightarrow S * S$
$S \rightarrow n$





Grammaire ambiguë

Exemple

$S \to S + S$

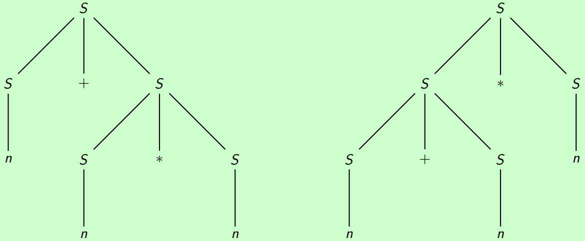
$S \to S * S$

$S \to n$

Dérivations de $n + n * n$:

$S \Rightarrow_g S + S \Rightarrow_g n + S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$

$S \Rightarrow_g S * S \Rightarrow_g S + S * S \Rightarrow_g n + S * S \Rightarrow_g n + n * S \Rightarrow_g n + n * n$



Une telle grammaire est dite **ambiguë**.

Reconnaître des grammaires hors contexte

	Langage	Reconnaisseur
Type 0	Récursivement énumérable	
	Récursif	
Type 1	Contextuel	
Type 2	Hors contexte	
Type 3	Régulier	Automate fini

Reconnaître des grammaires hors contexte

	Langage	Reconnaisseur
Type 0	Récursivement énumérable	Indécidable
	Récursif	
Type 1	Contextuel	
Type 2	Hors contexte	
Type 3	Régulier	Automate fini

Reconnaître des grammaires hors contexte

	Langage	**Reconnaisseur**
Type 0	Récursivement énumérable	Indécidable
	Récursif	Machine de Turing *totale*
Type 1	Contextuel	
Type 2	Hors contexte	
Type 3	Régulier	Automate fini

Reconnaître des grammaires hors contexte

	Langage	**Reconnaisseur**
Type 0	Récursivement énumérable	Indécidable
	Récursif	Machine de Turing *totale*
Type 1	Contextuel	MT linéairement bornée
Type 2	Hors contexte	
Type 3	Régulier	Automate fini

Reconnaître des grammaires hors contexte

	Langage	**Reconnaisseur**
Type 0	Récursivement énumérable	Indécidable
	Récursif	Machine de Turing *totale*
Type 1	Contextuel	MT linéairement bornée
Type 2	Hors contexte	**Automate à pile**
Type 3	Régulier	Automate fini

Automate à pile

Définition (Automate à pile)

Un **automate à pile** *(Pushdown automaton) est un 6-uplet* $(Q, \Sigma, \Gamma, \delta, q_0, F)$*, avec :*

- Q *est un ensemble fini d'états ;*
- Σ *est l'alphabet d'entrée ;*
- Γ *est l'***alphabet de pile** *;*
- $\delta : Q \times \Sigma \cup \{\epsilon\} \times \Gamma \cup \{\epsilon\} \rightarrow 2^{Q \times \Gamma \cup \{\epsilon\}}$ *est la fonction de transition ;*
- $q_0 \in Q$ *est l'état initial ;*
- $F \subseteq Q$ *est l'ensemble des états accepteurs.*

Automate à pile : configurations

- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :

Automate à pile : configurations

- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;

Automate à pile : configurations

- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;
  - $s \in \Gamma^*$ est le contenu de la pile.

Automate à pile : configurations

- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;
  - $s \in \Gamma^*$ est le contenu de la pile.
- La configuration **initiale** est (q_0, ϵ) ;

Automate à pile : configurations

- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;
  - $s \in \Gamma^*$ est le contenu de la pile. **mémoire initiale**
- La configuration **initiale** est (q_0, ϵ) ;
- Sur lecture de $a \in \Sigma$, l'automate passe de la configuration $(q, s\alpha)$ à la configuration $(q', s\beta)$, noté $(q, s\alpha) \xrightarrow{a} (q', s\beta)$ si :

Automate à pile : configurations

- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;
  - $s \in \Gamma^*$ est le contenu de la pile.
- La configuration **initiale** est (q_0, ϵ) ;
- Sur lecture de $a \in \Sigma$, l'automate passe de la configuration $(q, s\alpha)$ à la configuration $(q', s\beta)$, noté $(q, s\alpha) \xrightarrow{a} (q', s\beta)$ si :
  - $\alpha, \beta \in \Gamma$;

Automate à pile : configurations

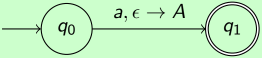
- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;
  - $s \in \Gamma^*$ est le contenu de la pile.
- La configuration **initiale** est (q_0, ϵ) ;
- Sur lecture de $a \in \Sigma$, l'automate passe de la configuration $(q, s\alpha)$ à la configuration $(q', s\beta)$, noté $(q, s\alpha) \xrightarrow{a} (q', s\beta)$ si :
  - $\alpha, \beta \in \Gamma$;
  - $(q', \beta) \in \delta(q, a, \alpha)$.

Automate à pile : configurations

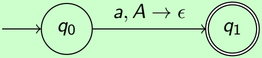
- Une **configuration** d'un automate à pile $(Q, \Sigma, \Gamma, \delta, q_0, F)$ est un couple (q, s) où :
  - $q \in Q$ est un état de l'automate ;
  - $s \in \Gamma^*$ est le contenu de la pile.
- La configuration **initiale** est (q_0, ϵ) ;
- Sur lecture de $a \in \Sigma$, l'automate passe de la configuration $(q, s\alpha)$ à la configuration $(q', s\beta)$, noté $(q, s\alpha) \xrightarrow{a} (q', s\beta)$ si :
  - $\alpha, \beta \in \Gamma$;
  - $(q', \beta) \in \delta(q, a, \alpha)$.
- $w \in \Sigma^*$ est **accepté** par l'automate si $(q_0, \epsilon) \xrightarrow{w}^* (q, s)$ avec $q \in F$
  $\rightarrow^*$=Image par la fermeture réflexive transitive de $\rightarrow$
  On peut aussi se passer de F et accepter quand la pile est vide

Automate à pile : exemples

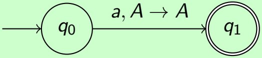
Exemple



Lit a et **empile** A



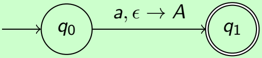
Lit a et **dépile** A



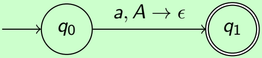
Lit a et **teste** A

Automate à pile : exemples

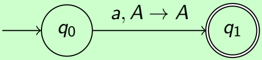
Exemple



Lit a et **empile** A

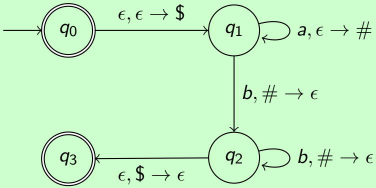


Lit a et **dépile** A



Lit a et **teste** A

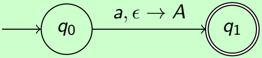
Exemple



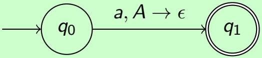
$L =$

Automate à pile : exemples

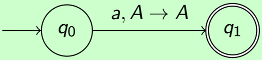
Exemple



Lit a et **empile** A

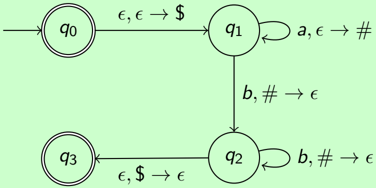


Lit a et **dépile** A



Lit a et **teste** A

Exemple

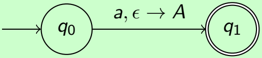


$$L = \{a^n b^n | n \in \mathbb{N}\}$$

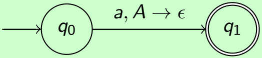
Si on arrive à q3 il y plus de b que a , encore de b à lire —> blocage

Automate à pile : exemples

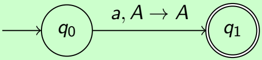
Exemple



Lit a et **empile** A

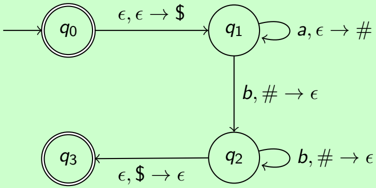


Lit a et **dépile** A



Lit a et **teste** A

Exemple



$L = \{a^n b^n | n \in \mathbb{N}\}$

Exercice

Donner un automate à pile qui reconnaît $L = \{wcw^R | w \in \{a, b\}^*\}$.

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \rightarrow W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)
  - Pour retrouver un automate sans transitions de lecture on duplique tous les états pour chaque lettre de l'alphabet (plus ϵ)
    s_a est l'état s avec la prochaine lettre à lire a (ϵ signifie qu'il n'y a plus rien à lire)

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)
  - Pour retrouver un automate sans transitions de lecture on duplique tous les états pour chaque lettre de l'alphabet (plus ϵ)
    s_a est l'état s avec la prochaine lettre à lire a (ϵ signifie qu'il n'y a plus rien à lire)
  - Les transitions (de lecture ou non) sont possibles seulement depuis la version de l'état correspondant à la lettre lue (aucune transition n'est possible depuis les états correspondant à ϵ) ;

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)
  - Pour retrouver un automate sans transitions de lecture on duplique tous les états pour chaque lettre de l'alphabet (plus ϵ)
    s_a est l'état s avec la prochaine lettre à lire a (ϵ signifie qu'il n'y a plus rien à lire)
  - Les transitions (de lecture ou non) sont possibles seulement depuis la version de l'état correspondant à la lettre lue (aucune transition n'est possible depuis les états correspondant à ϵ) ;
  - Les lectures mènent à l'état cible avec le même indice que la source ;

Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)
  - Pour retrouver un automate sans transitions de lecture on duplique tous les états pour chaque lettre de l'alphabet (plus ϵ)
    s_a est l'état s avec la prochaine lettre à lire a (ϵ signifie qu'il n'y a plus rien à lire)
  - Les transitions (de lecture ou non) sont possibles seulement depuis la version de l'état correspondant à la lettre lue (aucune transition n'est possible depuis les états correspondant à ϵ) ;
  - Les lectures mènent à l'état cible avec le même indice que la source ;
  - Les consommations mènent à toutes les versions de l'état cible ;

Automate (à pile) : transitions sans consommation

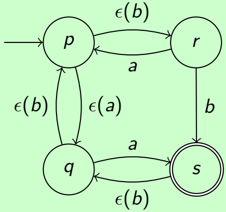
- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)
  - Pour retrouver un automate sans transitions de lecture on duplique tous les états pour chaque lettre de l'alphabet (plus ϵ)
    s_a est l'état s avec la prochaine lettre à lire a (ϵ signifie qu'il n'y a plus rien à lire)
  - Les transitions (de lecture ou non) sont possibles seulement depuis la version de l'état correspondant à la lettre lue (aucune transition n'est possible depuis les états correspondant à ϵ) ;
  - Les lectures mènent à l'état cible avec le même indice que la source ;
  - Les consommations mènent à toutes les versions de l'état cible ;
  - Un état s_ϵ est accepteur ssi s est accepteur ;

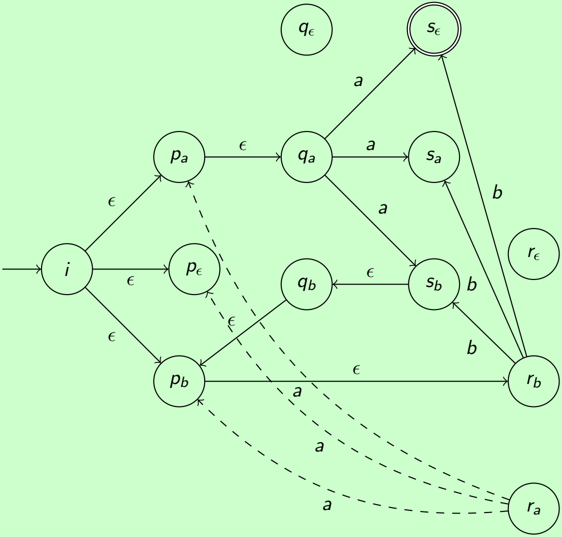
Automate (à pile) : transitions sans consommation

- On peut **étendre** facilement les automates (à pile ou non) pour prendre en compte des transitions qui **lisent** la lettre courante **sans la consommer**. On notera $q \xrightarrow{\epsilon(a), W \to W'} q'$ une telle transition ;
- L'**implémentation** de cette extension est triviale ;
- Cela ne change pas l'**expressivité** du modèle en termes de langage (dans le cas non-déterministe)
  - Pour retrouver un automate sans transitions de lecture on duplique tous les états pour chaque lettre de l'alphabet (plus ϵ)
    s_a est l'état s avec la prochaine lettre à lire a (ϵ signifie qu'il n'y a plus rien à lire)
  - Les transitions (de lecture ou non) sont possibles seulement depuis la version de l'état correspondant à la lettre lue (aucune transition n'est possible depuis les états correspondant à ϵ) ;
  - Les lectures mènent à l'état cible avec le même indice que la source ;
  - Les consommations mènent à toutes les versions de l'état cible ;
  - Un état s_ϵ est accepteur ssi s est accepteur ;
  - Les états initiaux sont les différentes versions dupliquées des états initiaux d'origine.

Automate (à pile) : transitions sans consommation

Exemple





Construction d'un analyseur

- Pour construire **automatiquement** un automate à pile reconnaissant une grammaire donnée :
  - méthode **descendante** : on part de l'axiome et on dérive jusqu'à trouver la chaîne ;
  - méthode **ascendante** : on part de la chaîne et on remonte les dérivations possibles jusqu'à trouver l'axiome ;

Construction d'un analyseur déterministe

- Simuler un automate à pile non déterministe est trop coûteux.

Construction d'un analyseur déterministe

- Simuler un automate à pile non déterministe est trop coûteux.
- On veut un automate **déterministe** : pour tous q, a, W :
  $\delta(q, \epsilon, W) = \emptyset$ et $\{(q', W'), (q'', W'')\} \subseteq \delta(q, a, W) \Rightarrow (q', W') = (q'', W'')$
  Idem pour les transitions sans consommation

Construction d'un analyseur déterministe

- Simuler un automate à pile non déterministe est trop coûteux.
- On veut un automate **déterministe** : pour tous q, a, W :
  $\delta(q, \epsilon, W) = \emptyset$ et $\{(q', W'), (q'', W'')\} \subseteq \delta(q, a, W) \Rightarrow (q', W') = (q'', W'')$
  Idem pour les transitions sans consommation
- **mais** :

Théorème

Les langages reconnus par les automates à pile ***déterministes*** *forment un sous-ensemble* ***strict*** *des langages hors contexte.*

p.ex. le langage $\{ww^R | w \in \{a, b\}^*\}$ ne peut pas être reconnu par un automate à pile déterministe

Construction d'un analyseur déterministe

- Simuler un automate à pile non déterministe est trop coûteux.
- On veut un automate **déterministe** : pour tous q, a, W :
  $\delta(q, \epsilon, W) = \emptyset$ et $\{(q', W'), (q'', W'')\} \subseteq \delta(q, a, W) \Rightarrow (q', W') = (q'', W'')$
  Idem pour les transitions sans consommation
- **mais** :

Théorème

Les langages reconnus par les automates à pile **déterministes** *forment un sous-ensemble* **strict** *des langages hors contexte.*

p.ex. le langage $\{ww^R | w \in \{a, b\}^*\}$ ne peut pas être reconnu par un automate à pile déterministe

- On propose des constructions d'automates **déterministes** pour des **sous-ensembles** des langages hors contexte déterministes.
  Mais qui permettent de reconnaître la très grande majorité des langages intéressants en pratique.

Plan

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

- Trois états : $\{\mathcal{I}, \mathcal{C}, \mathcal{F}\}$;

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

- Trois états : $\{\mathcal{I}, \mathcal{C}, \mathcal{F}\}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

- Trois états : $\{\mathcal{I}, \mathcal{C}, \mathcal{F}\}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle $A \rightarrow W$ de la grammaire, on ajoute une transition :
  En étendant la définition des automates pour pouvoir empiler des mots : empiler ABC c'est empiler C, puis B, puis A

$$\delta(\mathcal{C}, \epsilon, A) = \{(\mathcal{C}, W)\}$$

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

- Trois états : $\{\mathcal{I}, \mathcal{C}, \mathcal{F}\}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle $A \rightarrow W$ de la grammaire, on ajoute une transition :
  En étendant la définition des automates pour pouvoir empiler des mots : empiler ABC c'est empiler C, puis B, puis A

$$\delta(\mathcal{C}, \epsilon, A) = \{(\mathcal{C}, W)\}$$

- Pour tout terminal a, on ajoute une transition :

$$\delta(\mathcal{C}, a, a) = \{(\mathcal{C}, \epsilon)\}$$

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

- Trois états : $\{\mathcal{I}, \mathcal{C}, \mathcal{F}\}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle $A \to W$ de la grammaire, on ajoute une transition :
  En étendant la définition des automates pour pouvoir empiler des mots : empiler ABC c'est empiler C, puis B, puis A

$$\delta(\mathcal{C}, \epsilon, A) = \{(\mathcal{C}, W)\}$$

- Pour tout terminal a, on ajoute une transition :

$$\delta(\mathcal{C}, a, a) = \{(\mathcal{C}, \epsilon)\}$$

- On ajoute la transition d'initialisation :

$$\delta(\mathcal{I}, \epsilon, \epsilon) = \{(\mathcal{C}, \$S)\}$$

Automate à pile non déterministe descendant

On construit **toutes** les dérivations gauche possibles de l'axiome :

- Trois états : $\{\mathcal{I}, \mathcal{C}, \mathcal{F}\}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle $A \to W$ de la grammaire, on ajoute une transition :
  En étendant la définition des automates pour pouvoir empiler des mots : empiler ABC c'est empiler C, puis B, puis A

  $$\delta(\mathcal{C}, \epsilon, A) = \{(\mathcal{C}, W)\}$$

- Pour tout terminal a, on ajoute une transition :

  $$\delta(\mathcal{C}, a, a) = \{(\mathcal{C}, \epsilon)\}$$

- On ajoute la transition d'initialisation :

  $$\delta(\mathcal{I}, \epsilon, \epsilon) = \{(\mathcal{C}, \$S)\}$$

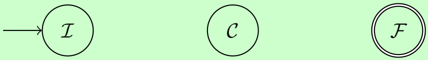
- On ajoute la transition d'acceptation :
  On suppose la chaîne à lire terminée par un \$.

  $$\delta(\mathcal{C}, \$, \$) = \{(\mathcal{F}, \epsilon)\}$$

Automate à pile non déterministe descendant

Exemple

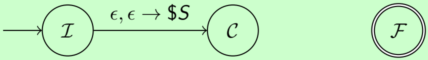
$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$



Automate à pile non déterministe descendant

Exemple

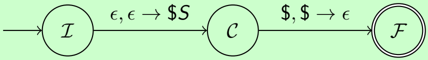
$S \to aSa$
$S \to bSb$
$S \to c$



Automate à pile non déterministe descendant

Exemple

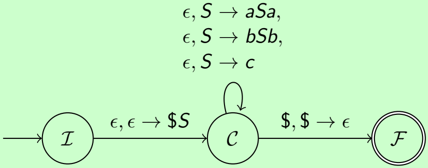
$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$



Automate à pile non déterministe descendant

Exemple

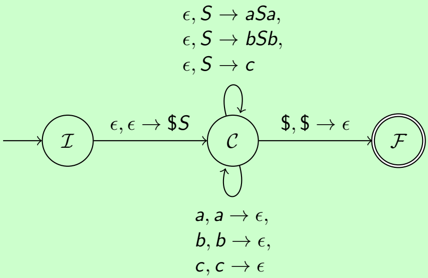
$S \to aSa$
$S \to bSb$
$S \to c$



Automate à pile non déterministe descendant

Exemple

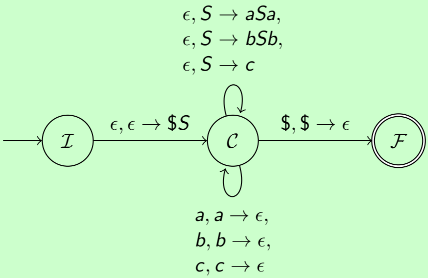
$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$



Automate à pile non déterministe descendant

Exemple

$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$



Exemple d'**exécution** de l'automate sur *abcba*\$:

$$(\mathcal{I}, \epsilon) \xrightarrow{\epsilon} (\mathcal{C}, \$S) \xrightarrow{\epsilon} (\mathcal{C}, \$aSa) \xrightarrow{a} (\mathcal{C}, \$aS) \xrightarrow{\epsilon} (\mathcal{C}, \$abSb) \xrightarrow{b} (\mathcal{C}, \$abS) \xrightarrow{\epsilon} (\mathcal{C}, \$abc)$$
$$\xrightarrow{c} (\mathcal{C}, \$ab) \xrightarrow{b} (\mathcal{C}, \$a) \xrightarrow{a} (\mathcal{C}, \$) \xrightarrow{\$} (\mathcal{F}, \epsilon)$$

Ensembles PREMIER et SUIVANT

- La construction précédente n'a (quasiment) aucune chance de donner un automate **déterministe** ;

Ensembles PREMIER et SUIVANT

- La construction précédente n'a (quasiment) aucune chance de donner un automate **déterministe** ;
- On va raffiner et restreignant les règles qui peuvent s'appliquer en fonction du **prochain** caractère à lire ;

Ensembles PREMIER et SUIVANT

- La construction précédente n'a (quasiment) aucune chance de donner un automate **déterministe** ;
- On va raffiner et restreignant les règles qui peuvent s'appliquer en fonction du **prochain** caractère à lire ;
- On construit pour cela deux ensembles :

Ensembles PREMIER et SUIVANT

- La construction précédente n'a (quasiment) aucune chance de donner un automate **déterministe** ;
- On va raffiner et restreignant les règles qui peuvent s'appliquer en fonction du **prochain** caractère à lire ;
- On construit pour cela deux ensembles :
  - PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$;

Ensembles PREMIER et SUIVANT

- La construction précédente n'a (quasiment) aucune chance de donner un automate **déterministe** ;
- On va raffiner et restreignant les règles qui peuvent s'appliquer en fonction du **prochain** caractère à lire ;
- On construit pour cela deux ensembles :
  - PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$;
  - SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \ldots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \dots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Exemple

$\Sigma = \{n, +, *, (,)\}$

$S \to TS'$	PREMIER(S) $=$
$S' \to +TS' \mid \epsilon$	PREMIER(S') $=$
$T \to FT'$	PREMIER(T) $=$
$T' \to *FT' \mid \epsilon$	PREMIER(T') $=$
$F \to (S) \mid n$	PREMIER(F) $=$

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \ldots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Exemple

$\Sigma = \{n, +, *, (,)\}$

$S \to TS'$	PREMIER(S) $=$
$S' \to +TS' \vert \epsilon$	PREMIER(S') $=$
$T \to FT'$	PREMIER(T) $=$
$T' \to *FT' \vert \epsilon$	PREMIER(T') $=$
$F \to (S) \vert n$	PREMIER(F) $= \{(, n\}$

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \ldots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Exemple

$S \to TS'$	PREMIER(S) $=$	$\Sigma = \{n, +, *, (,)\}$
$S' \to +TS' \mid \epsilon$	PREMIER(S') $= \{+, \epsilon\}$	
$T \to FT'$	PREMIER(T) $=$	
$T' \to *FT' \mid \epsilon$	PREMIER(T') $= \{*, \epsilon\}$	
$F \to (S) \mid n$	PREMIER(F) $= \{(, n\}$	

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \ldots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Exemple

$S \to TS'$	PREMIER(S) $=$	$\Sigma = \{n, +, *, (,)\}$
$S' \to +TS' \vert \epsilon$	PREMIER(S') $= \{+, \epsilon\}$	
$T \to FT'$	PREMIER(T) $=$ PREMIER(F)	$\epsilon \notin$ PREMIER(F)
$T' \to *FT' \vert \epsilon$	PREMIER(T') $= \{*, \epsilon\}$	
$F \to (S) \vert n$	PREMIER(F) $= \{(, n\}$	

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \ldots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Exemple

$\Sigma = \{n, +, *, (,)\}$

$S \to TS'$	$\text{PREMIER}(S) =$	
$S' \to +TS' \mid \epsilon$	$\text{PREMIER}(S') = \{+, \epsilon\}$	
$T \to FT'$	$\text{PREMIER}(T) = \text{PREMIER}(F) = \{(, n\}$	$\epsilon \notin \text{PREMIER}(F)$
$T' \to *FT' \mid \epsilon$	$\text{PREMIER}(T') = \{*, \epsilon\}$	
$F \to (S) \mid n$	$\text{PREMIER}(F) = \{(, n\}$	

Calcul de PREMIER

- PREMIER(W) est l'ensemble des **terminaux** qui commencent les chaînes dérivées de $W \in (\Sigma \cup N)^*$:

$$a \in \text{PREMIER}(W) \text{ ssi } \exists U \in (\Sigma \cup N)^* \text{ t.q. } W \Rightarrow_g^* aU$$

- Algorithme :
  1. Si X est un terminal, PREMIER(X) $= \{X\}$;
  2. Si $X \to \epsilon$ est une production, $\epsilon \in$ PREMIER(X) ;
  3. Si X est un non-terminal et $X \to W$ est une production alors PREMIER(W) $\subseteq$ PREMIER(X)
  4. $a \in$ PREMIER($Y_1 Y_2 \ldots Y_k$) s'il existe $i \leq k$ t.q. $a \in$ PREMIER(Y_i) et pour tout $j < i$, $\epsilon \in$ PREMIER(Y_j).

Exemple

$S \to TS'$	PREMIER(S) = PREMIER(T) = $\{(, n\}$	$\Sigma = \{n, +, *, (,)\}$
$S' \to +TS' \mid \epsilon$	PREMIER(S') = $\{+, \epsilon\}$	
$T \to FT'$	PREMIER(T) = PREMIER(F) = $\{(, n\}$	$\epsilon \notin$ PREMIER(F)
$T' \to *FT' \mid \epsilon$	PREMIER(T') = $\{*, \epsilon\}$	
$F \to (S) \mid n$	PREMIER(F) = $\{(, n\}$	

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \rightarrow UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \rightarrow UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \rightarrow UB$ ou $A \rightarrow UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \to UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \to UB$ ou $A \to UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$S \to TS'$
$S' \to +TS'|\epsilon$
$T \to FT'$
$T' \to *FT'|\epsilon$
$F \to (S)|n$

SUIVANT(S) = {\$,   (1),
SUIVANT(S') =
SUIVANT(T) =
SUIVANT(T') =
SUIVANT(F) =

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \to UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \to UB$ ou $A \to UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$S \to TS'$
$S' \to +TS'|\epsilon$
$T \to FT'$
$T' \to *FT'|\epsilon$
$F \to (S)|n$

SUIVANT(S) $= \{\$,)\}$ (1), (2)
SUIVANT(S') $=$
SUIVANT(T) $=$
SUIVANT(T') $=$
SUIVANT(F) $=$

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \to UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \to UB$ ou $A \to UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$S \to TS'$	SUIVANT(S) $= \{\$,)\}$	(1), (2)
$S' \to +TS' \vert \epsilon$	SUIVANT(S') $=$	
$T \to FT'$	SUIVANT(T) $= \{+,$	(2)
$T' \to *FT' \vert \epsilon$	SUIVANT(T') $=$	
$F \to (S) \vert n$	SUIVANT(F) $= \{*,$	(2)

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \rightarrow UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \rightarrow UB$ ou $A \rightarrow UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$S \rightarrow TS'$	$\text{SUIVANT}(S) = \{\$,)\}$	(1), (2)
$S' \rightarrow +TS' \mid \epsilon$	$\text{SUIVANT}(S') = \{\$,)\}$	(3)
$T \rightarrow FT'$	$\text{SUIVANT}(T) = \{+,$	(2)
$T' \rightarrow *FT' \mid \epsilon$	$\text{SUIVANT}(T') =$	
$F \rightarrow (S) \mid n$	$\text{SUIVANT}(F) = \{*,$	(2)

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \rightarrow UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \rightarrow UB$ ou $A \rightarrow UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$S \rightarrow TS'$
$S' \rightarrow +TS'|\epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT'|\epsilon$
$F \rightarrow (S)|n$

SUIVANT(S) $= \{\$,)\}$	(1), (2)
SUIVANT(S') $= \{\$,)\}$	(3)
SUIVANT(T) $= \{+, \$,)\}$	(2), (3ϵ)
SUIVANT(T') $=$	
SUIVANT(F) $= \{*,$	(2)

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \rightarrow UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \rightarrow UB$ ou $A \rightarrow UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$$S \rightarrow TS'$$
$$S' \rightarrow +TS'|\epsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT'|\epsilon$$
$$F \rightarrow (S)|n$$

$\text{SUIVANT}(S) = \{\$,)\}$	$(1), (2)$
$\text{SUIVANT}(S') = \{\$,)\}$	(3)
$\text{SUIVANT}(T) = \{+, \$,)\}$	$(2), (3\epsilon)$
$\text{SUIVANT}(T') = \{+, \$,)\}$	(3)
$\text{SUIVANT}(F) = \{*,$	(2)

Calcul de SUIVANT

- SUIVANT(A) est l'ensemble des **terminaux** qui peuvent apparaître à droite du non-terminal $A \in N$:

$$a \in \text{SUIVANT}(A) \text{ ssi } \exists U, V \in (\Sigma \cup N)^* \text{ t.q. } S \Rightarrow_g^* UAaV$$

- Algorithme :
  1. On ajoute un \$ à la fin de la chaîne à reconnaître et à SUIVANT(S) (S est l'axiome).
  2. Si $A \to UBV$ est une production, PREMIER(V) $\setminus \{\epsilon\}$ est inclus dans SUIVANT(B)
  3. Si $A \to UB$ ou $A \to UBV$ et $\epsilon \in$ PREMIER(V), alors SUIVANT(A) est inclus dans SUIVANT(B).

Exemple

$S \to TS'$	$\text{SUIVANT}(S) = \{\$,)\}$	$(1), (2)$
$S' \to +TS' \mid \epsilon$	$\text{SUIVANT}(S') = \{\$,)\}$	(3)
$T \to FT'$	$\text{SUIVANT}(T) = \{+, \$,)\}$	$(2), (3\epsilon)$
$T' \to *FT' \mid \epsilon$	$\text{SUIVANT}(T') = \{+, \$,)\}$	(3)
$F \to (S) \mid n$	$\text{SUIVANT}(F) = \{*, +, \$,)\}$	$(2), (3\epsilon)$

PREMIER et SUIVANT : Exercice

Exercice

Calculer les ensembles PREMIER et SUIVANT des grammaires suivantes :

$$G_1 = \begin{cases} S \rightarrow ibtSE|a \\ E \rightarrow eS|\epsilon \end{cases} \quad G_2 = \begin{cases} S \rightarrow ACB \\ A \rightarrow aA|b \\ B \rightarrow Bb|a \\ C \rightarrow aBcAb \end{cases} \quad G_3 = \begin{cases} S \rightarrow ACcaB \\ A \rightarrow aB|\epsilon \\ B \rightarrow Bb|c \\ C \rightarrow bAb|a \end{cases}$$

Analyseur prédictif

On construit l'automate en **restreignant** les substitutions :

- **États** : $\mathcal{I}$ (initial), $\mathcal{F}$ (final), $\mathcal{E}$, $\mathcal{C}$;

Analyseur prédictif

On construit l'automate en **restreignant** les substitutions :

- **États** : $\mathcal{I}$ (initial), $\mathcal{F}$ (final), $\mathcal{E}$, $\mathcal{C}$;
- Pour $A \rightarrow W$ et $a \in \Sigma$, **si** $a \in \text{PREMIER}(W)$, ou $a \in \text{SUIVANT}(A)$ et $\epsilon \in \text{PREMIER}(W)$, on ajoute une transition :

$$\delta(\mathcal{C}, \epsilon(\mathbf{a}), A) = \{(\mathcal{C}, W)\}$$

Analyseur prédictif

On construit l'automate en **restreignant** les substitutions :

- **États** : $\mathcal{I}$ (initial), $\mathcal{F}$ (final), $\mathcal{E}$, $\mathcal{C}$;
- Pour $A \to W$ et $a \in \Sigma$, **si** $a \in \text{PREMIER}(W)$, ou $a \in \text{SUIVANT}(A)$ et $\epsilon \in \text{PREMIER}(W)$, on ajoute une transition :

$$\delta(\mathcal{C}, \epsilon(\mathbf{a}), A) = \{(\mathcal{C}, W)\}$$

- Pour tout $a \in \Sigma \cup \{\$\}, b \in \Sigma$, on ajoute la transition :

$$\delta(\mathcal{C}, a, a) = \{(\mathcal{C}, \epsilon)\}$$

Analyseur prédictif

On construit l'automate en **restreignant** les substitutions :

- **États** : $\mathcal{I}$ (initial), $\mathcal{F}$ (final), $\mathcal{E}$, $\mathcal{C}$;
- Pour $A \to W$ et $a \in \Sigma$, **si** $a \in \text{PREMIER}(W)$, ou $a \in \text{SUIVANT}(A)$ et $\epsilon \in \text{PREMIER}(W)$, on ajoute une transition :

$$\delta(\mathcal{C}, \epsilon(\mathbf{a}), A) = \{(\mathcal{C}, W)\}$$

- Pour tout $a \in \Sigma \cup \{\$\}$, $b \in \Sigma$, on ajoute la transition :

$$\delta(\mathcal{C}, a, a) = \{(\mathcal{C}, \epsilon)\}$$

- Initialisation (quelle que soit la première lettre) et acceptation (on suppose la chaîne à lire terminée par \$) :

$$\delta(\mathcal{I}, \epsilon(\mathbf{\Sigma}), \epsilon) = \{(\mathcal{C}, \$S)\} \text{ et } \delta(\mathcal{C}, \$, \$) = \{(\mathcal{F}, \epsilon)\}$$

Analyseur prédictif

On construit l'automate en **restreignant** les substitutions :

- **États** : $\mathcal{I}$ (initial), $\mathcal{F}$ (final), $\mathcal{E}$, $\mathcal{C}$;
- Pour $A \to W$ et $a \in \Sigma$, **si** $a \in \text{PREMIER}(W)$, ou $a \in \text{SUIVANT}(A)$ et $\epsilon \in \text{PREMIER}(W)$, on ajoute une transition :

$$\delta(\mathcal{C}, \epsilon(\mathbf{a}), A) = \{(\mathcal{C}, W)\}$$

- Pour tout $a \in \Sigma \cup \{\$\}, b \in \Sigma$, on ajoute la transition :

$$\delta(\mathcal{C}, a, a) = \{(\mathcal{C}, \epsilon)\}$$

- Initialisation (quelle que soit la première lettre) et acceptation (on suppose la chaîne à lire terminée par \$) :

$$\delta(\mathcal{I}, \epsilon(\boldsymbol{\Sigma}), \epsilon) = \{(\mathcal{C}, \$S)\} \text{ et } \delta(\mathcal{C}, \$, \$) = \{(\mathcal{F}, \epsilon)\}$$

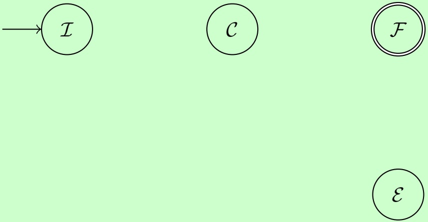
- On complète δ avec les transitions d'**erreur** : pour tout B t.q. $\nexists B \to W$ t.q. $a \in \text{PREMIER}(W)$,

$$\delta(\mathcal{C}, \epsilon(a), B) = \{(\mathcal{E}, \epsilon)\}$$

Analyseur prédictif

Exemple

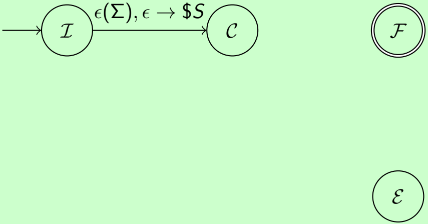
$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \to aSa$
$S \to bSb$
$S \to c$
$\text{PREMIER}(aSa) = \{a\}$
$\text{PREMIER}(bSb) = \{b\}$
$\text{PREMIER}(c) = \{c\}$



Analyseur prédictif

Exemple

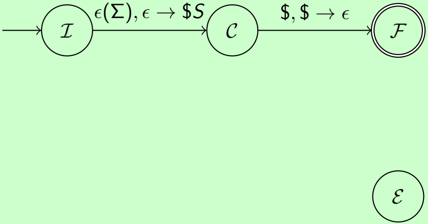
$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$
PREMIER(aSa) $= \{a\}$
PREMIER(bSb) $= \{b\}$
PREMIER(c) $= \{c\}$



Analyseur prédictif

Exemple

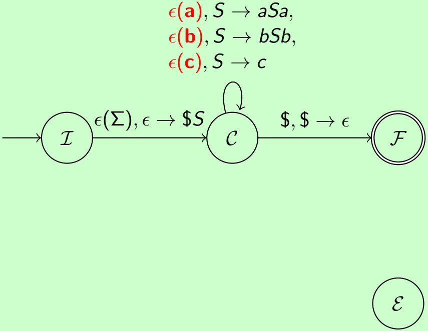
$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \to aSa$
$S \to bSb$
$S \to c$
$\text{PREMIER}(aSa) = \{a\}$
$\text{PREMIER}(bSb) = \{b\}$
$\text{PREMIER}(c) = \{c\}$



Analyseur prédictif

Exemple

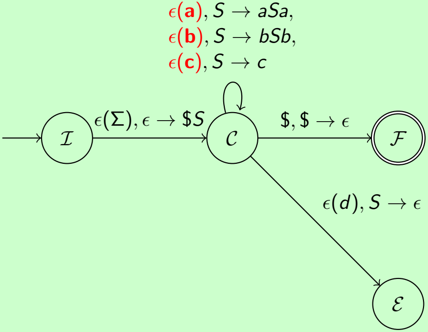
$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \to aSa$
$S \to bSb$
$S \to c$
PREMIER(aSa) $= \{a\}$
PREMIER(bSb) $= \{b\}$
PREMIER(c) $= \{c\}$



Analyseur prédictif

Exemple

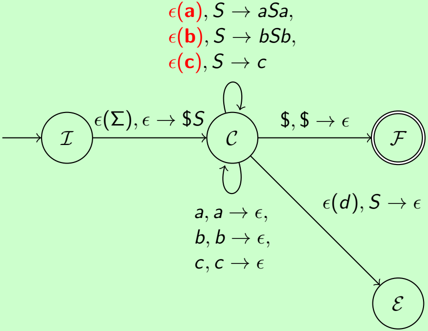
$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \to aSa$
$S \to bSb$
$S \to c$
PREMIER(aSa) $= \{a\}$
PREMIER(bSb) $= \{b\}$
PREMIER(c) $= \{c\}$



Analyseur prédictif

Exemple

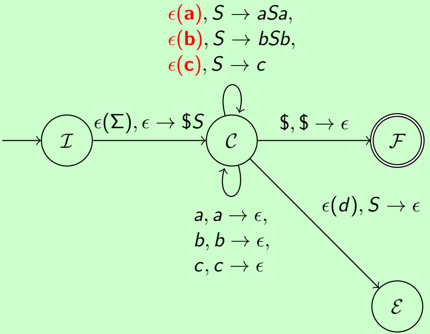
$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \to aSa$
$S \to bSb$
$S \to c$
PREMIER$(aSa) = \{a\}$
PREMIER$(bSb) = \{b\}$
PREMIER$(c) = \{c\}$



Analyseur prédictif

Exemple

$\Sigma = \{a, b, c, \mathbf{d}\}$
$S \to aSa$
$S \to bSb$
$S \to c$
PREMIER$(aSa) = \{a\}$
PREMIER$(bSb) = \{b\}$
PREMIER$(c) = \{c\}$



Exemple d'**exécution** de l'automate sur *abcba*\$:

$$(\mathcal{I}, \epsilon) \xrightarrow{\epsilon(a)} (\mathcal{C}, \$S) \xrightarrow{\epsilon(a)} (\mathcal{C}, \$aSa) \xrightarrow{a} (\mathcal{C}, \$aS) \xrightarrow{\epsilon(b)} (\mathcal{C}, \$abSb) \xrightarrow{b} (\mathcal{C}, \$abS)$$
$$\xrightarrow{\epsilon(c)} (\mathcal{C}, \$abc) \xrightarrow{c} (\mathcal{C}, \$ab) \xrightarrow{b} (\mathcal{C}, \$a) \xrightarrow{a} (\mathcal{C}, \$) \xrightarrow{\$} (\mathcal{F}, \epsilon)$$

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \to U|V$ avec PREMIER(U) $\cap$ PREMIER(V) $\neq \emptyset$;

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \rightarrow U|V$ avec $\text{PREMIER}(U) \cap \text{PREMIER}(V) \neq \emptyset$;
  - PREMIER/SUIVANT : $A \rightarrow U|V$ avec $\epsilon \in \text{PREMIER}(V)$ et $\text{PREMIER}(U) \cap \text{SUIVANT}(A) \neq \emptyset$;

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \rightarrow U|V$ avec PREMIER$(U) \cap$ PREMIER$(V) \neq \emptyset$;
  - PREMIER/SUIVANT : $A \rightarrow U|V$ avec $\epsilon \in$ PREMIER(V) et PREMIER$(U) \cap$ SUIVANT$(A) \neq \emptyset$;
- Ces conflits peuvent avoir différentes causes, en particulier :

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \rightarrow U|V$ avec $\text{PREMIER}(U) \cap \text{PREMIER}(V) \neq \emptyset$;
  - PREMIER/SUIVANT : $A \rightarrow U|V$ avec $\epsilon \in \text{PREMIER}(V)$ et $\text{PREMIER}(U) \cap \text{SUIVANT}(A) \neq \emptyset$;
- Ces conflits peuvent avoir différentes causes, en particulier :
  - Le **langage** n'est pas LL(1) ;

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \rightarrow U|V$ avec PREMIER$(U) \cap$ PREMIER$(V) \neq \emptyset$;
  - PREMIER/SUIVANT : $A \rightarrow U|V$ avec $\epsilon \in$ PREMIER(V) et PREMIER$(U) \cap$ SUIVANT$(A) \neq \emptyset$;
- Ces conflits peuvent avoir différentes causes, en particulier :
  - Le **langage** n'est pas LL(1) ;
  - La grammaire est **ambiguë** ;

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \to U|V$ avec PREMIER$(U) \cap$ PREMIER$(V) \neq \emptyset$;
  - PREMIER/SUIVANT : $A \to U|V$ avec $\epsilon \in$ PREMIER(V) et PREMIER$(U) \cap$ SUIVANT$(A) \neq \emptyset$;
- Ces conflits peuvent avoir différentes causes, en particulier :
  - Le **langage** n'est pas LL(1) ;
  - La grammaire est **ambiguë** ;
  - ou **récursive** à gauche.

Grammaires LL(1)

- Si l'automate obtenu est **déterministe**, on dit que la grammaire est **LL(1)** et l'analyse a réussi ;
- Sinon, deux types de **conflits** entre les règles sont possibles :
  - PREMIER/PREMIER : $A \to U|V$ avec PREMIER$(U) \cap$ PREMIER$(V) \neq \emptyset$;
  - PREMIER/SUIVANT : $A \to U|V$ avec $\epsilon \in$ PREMIER(V) et PREMIER$(U) \cap$ SUIVANT$(A) \neq \emptyset$;
- Ces conflits peuvent avoir différentes causes, en particulier :
  - Le **langage** n'est pas LL(1) ;
  - La grammaire est **ambiguë** ;
  - ou **récursive** à gauche.
- Dans les deux derniers cas, on peut essayer de **transformer** la grammaire pour la rendre analysable par cette méthode.

Transformations : ambiguités

- On peut **parfois** supprimer les ambiguités dans une grammaire ;

Transformations : ambiguités

- On peut **parfois** supprimer les ambiguités dans une grammaire ;
- Il faut imposer un **unique** arbre de dérivation pour chaque phrase :

Transformations : ambiguités

- On peut **parfois** supprimer les ambiguités dans une grammaire ;
- Il faut imposer un **unique** arbre de dérivation pour chaque phrase :

Exemple

$$S \to S + S$$
$$S \to S * S$$
$$S \to (S)|n$$

Transformations : ambiguités

- On peut **parfois** supprimer les ambiguités dans une grammaire ;
- Il faut imposer un **unique** arbre de dérivation pour chaque phrase :

Exemple

$$
\begin{aligned}
S &\rightarrow S + S\\
S &\rightarrow S * S\\
S &\rightarrow (S)|n
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow S + S\\
S &\rightarrow E\\
E &\rightarrow E * E\\
E &\rightarrow (S)|n
\end{aligned}
$$

Transformations : Grammaires propres

- Une grammaire (qui ne génère pas ϵ) est **propre** si elle n'a pas de production $A \to \epsilon$;

Transformations : Grammaires propres

- Une grammaire (qui ne génère pas ϵ) est **propre** si elle n'a pas de production $A \to \epsilon$;
- On peut **toujours** rendre une grammaire (qui ne génère pas ϵ) propre ;

Transformations : Grammaires propres

- Une grammaire (qui ne génère pas ϵ) est **propre** si elle n'a pas de production $A \to \epsilon$;
- On peut **toujours** rendre une grammaire (qui ne génère pas ϵ) propre ;
- Pour toute règle $A \to \epsilon$ et toute occurence de A dans une règle $B \to UAV$, on duplique cette dernière règle en remplaçant A par ϵ ($B \to UV$) :
  On fait toutes les possibilités

Transformations : Grammaires propres

- Une grammaire (qui ne génère pas ϵ) est **propre** si elle n'a pas de production $A \to \epsilon$;
- On peut **toujours** rendre une grammaire (qui ne génère pas ϵ) propre ;
- Pour toute règle $A \to \epsilon$ et toute occurence de A dans une règle $B \to UAV$, on duplique cette dernière règle en remplaçant A par ϵ ($B \to UV$) :
  On fait toutes les possibilités

Exemple

$$S \to aEcEb$$
$$E \to d|\epsilon$$

Transformations : Grammaires propres

- Une grammaire (qui ne génère pas ϵ) est **propre** si elle n'a pas de production $A \to \epsilon$;
- On peut **toujours** rendre une grammaire (qui ne génère pas ϵ) propre ;
- Pour toute règle $A \to \epsilon$ et toute occurence de A dans une règle $B \to UAV$, on duplique cette dernière règle en remplaçant A par ϵ ($B \to UV$) :
  On fait toutes les possibilités

Exemple

$$S \to aEcEb$$
$$E \to d|\epsilon$$

$$S \to aEcEb$$
$$S \to a\epsilon cEb$$
$$S \to aEc\epsilon b$$
$$S \to a\epsilon c\epsilon b$$
$$E \to d$$

Transformations : Grammaires propres

- Une grammaire (qui ne génère pas ϵ) est **propre** si elle n'a pas de production $A \to \epsilon$;
- On peut **toujours** rendre une grammaire (qui ne génère pas ϵ) propre ;
- Pour toute règle $A \to \epsilon$ et toute occurence de A dans une règle $B \to UAV$, on duplique cette dernière règle en remplaçant A par ϵ ($B \to UV$) :
  On fait toutes les possibilités

Exemple

$S \to aEcEb$ $E \to d \mid \epsilon$	$S \to aEcEb$ $S \to a\epsilon cEb$ $S \to aEc\epsilon b$ $S \to a\epsilon c\epsilon b$ $E \to d$	$S \to aEcEb$ $S \to acEb$ $S \to aEcb$ $S \to acb$ $E \to d$

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^{+} A$) non récursive à gauche :

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \rightarrow AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \rightarrow Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \rightarrow^{+} A$) non récursive à gauche :
  1. On remplace chaque règle $A \rightarrow BW$ par $A \rightarrow XW$ pour toute règle $B \rightarrow X$;

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^+ A$) non récursive à gauche :
  1. On remplace chaque règle $A \to BW$ par $A \to XW$ pour toute règle $B \to X$;
  2. On remplace chaque règle $A \to AU|V$ par $A \to VA'$ et $A' \to UA'|\epsilon$;

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^+ A$) non récursive à gauche :
  1. On remplace chaque règle $A \to BW$ par $A \to XW$ pour toute règle $B \to X$;
  2. On remplace chaque règle $A \to AU|V$ par $A \to VA'$ et $A' \to UA'|\epsilon$;
  3. On recommence jusqu'au point fixe.

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^+ A$) non récursive à gauche :
  1. On remplace chaque règle $A \to BW$ par $A \to XW$ pour toute règle $B \to X$;
  2. On remplace chaque règle $A \to AU|V$ par $A \to VA'$ et $A' \to UA'|\epsilon$;
  3. On recommence jusqu'au point fixe.

Exemple

$S \to Aa|b$
$A \to Ac|Sd|e$

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^+ A$) non récursive à gauche :
  1. On remplace chaque règle $A \to BW$ par $A \to XW$ pour toute règle $B \to X$;
  2. On remplace chaque règle $A \to AU|V$ par $A \to VA'$ et $A' \to UA'|\epsilon$;
  3. On recommence jusqu'au point fixe.

Exemple

$$\begin{array}{ll} & S \to Aa|b \\ S \to Aa|b & \mathbf{A \to SdA'|eA'} \\ A \to Ac|Sd|e & \mathbf{A' \to cA'|\epsilon} \end{array}$$

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^+ A$) non récursive à gauche :
  1. On remplace chaque règle $A \to BW$ par $A \to XW$ pour toute règle $B \to X$;
  2. On remplace chaque règle $A \to AU|V$ par $A \to VA'$ et $A' \to UA'|\epsilon$;
  3. On recommence jusqu'au point fixe.

Exemple

$S \to Aa|b$
$A \to Ac|Sd|e$

$S \to Aa|b$
$\mathbf{A \to SdA'|eA'}$
$\mathbf{A' \to cA'|\epsilon}$

$\mathbf{S \to SdA'a|eA'a|b}$
$A' \to cA'|\epsilon$

Transformations : Récursivité à gauche

- Une grammaire est **récursive à gauche** si elle a une production $A \to AU$;
- L'analyseur prédictif ne peut pas fonctionner avec une telle grammaire : Conflits PREMIER/PREMIER : p.ex. $A \to Aa|b$
- On peut **toujours** rendre une grammaire propre et sans cycle ($A \to^+ A$) non récursive à gauche :
  1. On remplace chaque règle $A \to BW$ par $A \to XW$ pour toute règle $B \to X$;
  2. On remplace chaque règle $A \to AU|V$ par $A \to VA'$ et $A' \to UA'|\epsilon$;
  3. On recommence jusqu'au point fixe.

Exemple

$$S \to Aa|b \qquad A \to Ac|Sd|e$$

$$S \to Aa|b \qquad \mathbf{A \to SdA'|eA'} \qquad \mathbf{A' \to cA'}|\epsilon$$

$$\mathbf{S \to SdA'a|eA'a|b} \qquad A' \to cA'|\epsilon$$

$$\mathbf{S \to eA'S'|bS'} \qquad \mathbf{S' \to dA'aS'}|\epsilon \qquad A' \to cA'|\epsilon$$

Plan

Analyse ascendante

- L'analyse descendante est limitée **en pratique**
(pour générer des automates déterministes)

Analyse ascendante

- L'analyse descendante est limitée **en pratique**
  (pour générer des automates déterministes)
- L'analyse **ascendante** permet de générer des analyseurs déterministes pour un plus grand nombre de grammaires.

Analyse ascendante

- L'analyse descendante est limitée **en pratique** (pour générer des automates déterministes)
- L'analyse **ascendante** permet de générer des analyseurs déterministes pour un plus grand nombre de grammaires.
- Elle se base sur le principe ≪ **décalage - réduction** ≫ (*shift* / *reduce*)

Analyse ascendante

- L'analyse descendante est limitée **en pratique**
  (pour générer des automates déterministes)
- L'analyse **ascendante** permet de générer des analyseurs déterministes pour un plus grand nombre de grammaires.
- Elle se base sur le principe « **décalage - réduction** » (*shift* / *reduce*)
- Cela consiste à **remonter**, à partir de la chaîne d'entrée, le long d'une dérivation **droite** possible.

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

- Les états : $\mathcal{I}$,$\mathcal{C}$,$\mathcal{B}$,$\mathcal{F}$;

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

- Les états : $\mathcal{I}$,$\mathcal{C}$,$\mathcal{B}$,$\mathcal{F}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

- Les états : $\mathcal{I}$,$\mathcal{C}$,$\mathcal{B}$,$\mathcal{F}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle R_i : $A \rightarrow B_1 \ldots B_k$, on ajoute (**reduce**) :
  En étendant l'automate pour dépiler plusieurs symboles successivement sur une transition

$$\delta(\mathcal{C}, \epsilon, B_1 \ldots B_k) = \{(\mathcal{C}, A)\}$$

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

- Les états : $\mathcal{I}$,$\mathcal{C}$,$\mathcal{B}$,$\mathcal{F}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle R_i : $A \rightarrow B_1 \dots B_k$, on ajoute (**reduce**) :
  En étendant l'automate pour dépiler plusieurs symboles successivement sur une transition

$$\delta(\mathcal{C}, \epsilon, B_1 \dots B_k) = \{(\mathcal{C}, A)\}$$

- Pour tout terminal a, on ajoute une transition (**shift**) :

$$\delta(\mathcal{C}, a, \epsilon) = \{(\mathcal{C}, a)\}$$

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

- Les états : $\mathcal{I}$,$\mathcal{C}$,$\mathcal{B}$,$\mathcal{F}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle R_i : $A \rightarrow B_1 \dots B_k$, on ajoute (**reduce**) :
  En étendant l'automate pour dépiler plusieurs symboles successivement sur une transition

$$\delta(\mathcal{C}, \epsilon, B_1 \dots B_k) = \{(\mathcal{C}, A)\}$$

- Pour tout terminal a, on ajoute une transition (**shift**) :

$$\delta(\mathcal{C}, a, \epsilon) = \{(\mathcal{C}, a)\}$$

- On ajoute la transition d'initialisation :

$$\delta(\mathcal{I}, \epsilon, \epsilon) = \{(\mathcal{C}, \$)\}$$

Automate à pile non déterministe ascendant

On construit **toutes** les dérivations droite inverses possibles de la chaîne d'entrée :

- Les états : $\mathcal{I}$,$\mathcal{C}$,$\mathcal{B}$,$\mathcal{F}$;
- L'état initial est $\mathcal{I}$, l'état accepteur $\mathcal{F}$;
- Pour toute règle $R_i : A \rightarrow B_1 \ldots B_k$, on ajoute (**reduce**) :
  En étendant l'automate pour dépiler plusieurs symboles successivement sur une transition
  $$\delta(\mathcal{C}, \epsilon, B_1 \ldots B_k) = \{(\mathcal{C}, A)\}$$
- Pour tout terminal a, on ajoute une transition (**shift**) :
  $$\delta(\mathcal{C}, a, \epsilon) = \{(\mathcal{C}, a)\}$$
- On ajoute la transition d'initialisation :
  $$\delta(\mathcal{I}, \epsilon, \epsilon) = \{(\mathcal{C}, \$)\}$$
- On ajoute les transitions d'acceptation :
  $$\delta(\mathcal{C}, \epsilon, S) = \{(\mathcal{B}, \epsilon)\} \text{ et } \delta(\mathcal{B}, \$, \$) = \{(\mathcal{F}, \epsilon)\}$$

Automate à pile non déterministe ascendant

Exemple

$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$

Automate à pile non déterministe ascendant

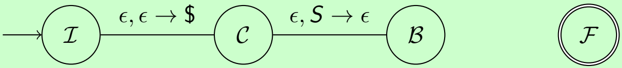
Exemple

$S \to aSa$
$S \to bSb$
$S \to c$

Automate à pile non déterministe ascendant

Exemple

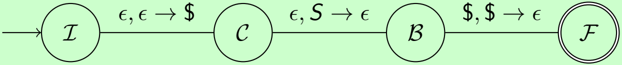
$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$



Automate à pile non déterministe ascendant

Exemple

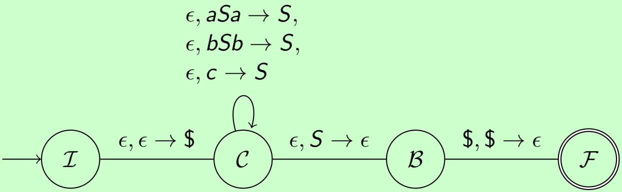
$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$



Automate à pile non déterministe ascendant

Exemple

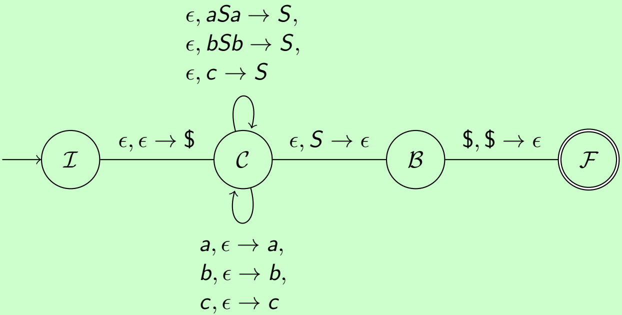
$S \to aSa$
$S \to bSb$
$S \to c$



Automate à pile non déterministe ascendant

Exemple

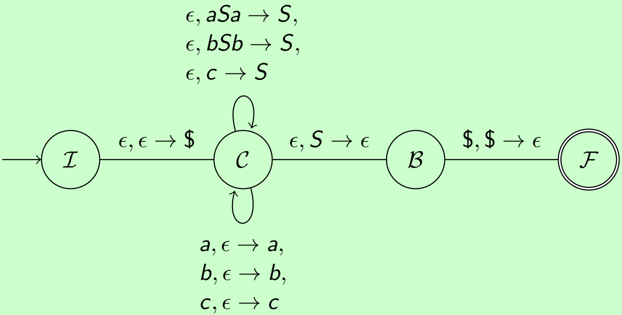
$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$



Automate à pile non déterministe ascendant

Exemple

$$S \to aSa$$
$$S \to bSb$$
$$S \to c$$



Exemple d'**exécution** de l'automate sur *abcba*\$:

$$(\mathcal{I}, \epsilon) \xrightarrow{\epsilon} (\mathcal{C}, \$) \xrightarrow{a} (\mathcal{C}, \$a) \xrightarrow{b} (\mathcal{C}, \$ab) \xrightarrow{c} (\mathcal{C}, \$abc) \xrightarrow{\epsilon} (\mathcal{C}, \$abS)$$
$$\xrightarrow{b} (\mathcal{C}, \$abSb) \xrightarrow{\epsilon} (\mathcal{C}, \$aS) \xrightarrow{a} (\mathcal{C}, \$aSa) \xrightarrow{\epsilon} (\mathcal{C}, \$S) \xrightarrow{\epsilon} (\mathcal{B}, \$) \xrightarrow{\$} (\mathcal{F}, \epsilon)$$

Déterminisation : conflits

- Comme précédemment, l'automate a peu de chance d'être déterministe ;

Déterminisation : conflits

- Comme précédemment, l'automate a peu de chance d'être déterministe ;
- Deux types de choix non déterministes (**conflits**) peuvent survenir :

Déterminisation : conflits

- Comme précédemment, l'automate a peu de chance d'être déterministe ;
- Deux types de choix non déterministes (**conflits**) peuvent survenir :
    1. décalage - réduction (*shift-reduce*) ;

Déterminisation : conflits

- Comme précédemment, l'automate a peu de chance d'être déterministe ;
- Deux types de choix non déterministes (**conflits**) peuvent survenir :
  1. décalage - réduction (*shift-reduce*) ;
  2. réduction - réduction (*reduce-reduce*) ;

Conflit décalage - réduction

Il a un conflit **décalage - réduction** quand on peut aussi bien :

- décaler sur la pile le symbole lu ;

Conflit décalage - réduction

Il a un conflit **décalage - réduction** quand on peut aussi bien :

- décaler sur la pile le symbole lu ;
- ou réduire le haut de la pile par une règle (inverse) de la grammaire.

Conflit décalage - réduction

Il a un conflit **décalage - réduction** quand on peut aussi bien :

- décaler sur la pile le symbole lu ;
- ou réduire le haut de la pile par une règle (inverse) de la grammaire.

Exemple

$R_1 : S \rightarrow \textbf{si}E\textbf{alors}S$
$R_2 : S \rightarrow \textbf{si}E\textbf{alors}S\textbf{sinon}S$
$\ldots$

État de l'automate :
$(\mathcal{C}, \$ \ldots \textbf{si}E\textbf{alors}S)$

Entrée :
$\textbf{sinon} \ldots \$$

Conflit décalage - réduction

Il a un conflit **décalage - réduction** quand on peut aussi bien :

- décaler sur la pile le symbole lu ;
- ou réduire le haut de la pile par une règle (inverse) de la grammaire.

Exemple

$R_1 : S \rightarrow \textbf{si}E\textbf{alors}S$	État de l'automate :	Entrée :
$R_2 : S \rightarrow \textbf{si}E\textbf{alors}S\textbf{sinon}S$	$(\mathcal{C}, \$ \ldots \textbf{si}E\textbf{alors}S)$	$\textbf{sinon} \ldots \$$
$\ldots$		

Réduire par R_1 ou décaler puis réduire par R_2 ?

Conflit réduction - réduction

Il a un conflit **réduction - réduction** quand on peut aussi bien :

- réduire le haut de la pile par une règle R_1.

Conflit réduction - réduction

Il a un conflit **réduction - réduction** quand on peut aussi bien :

- réduire le haut de la pile par une règle R_1.
- ou réduire le haut de la pile par une règle $R_2 \neq R_1$.

Conflit réduction - réduction

Il a un conflit **réduction - réduction** quand on peut aussi bien :

- réduire le haut de la pile par une règle R_1.
- ou réduire le haut de la pile par une règle $R_2 \neq R_1$.

Exemple

R_1 : $Instr \to \textbf{nom}(Expr, Expr)$	État de l'automate :	Entrée :
R_2 : $Expr \to \textbf{nom}(Expr, Expr)$	$(\mathcal{C}, \$ \ldots \textbf{nom}(Expr, Expr))$	$\ldots \$$
$\ldots$		

Conflit réduction - réduction

Il a un conflit **réduction - réduction** quand on peut aussi bien :

- réduire le haut de la pile par une règle R_1.
- ou réduire le haut de la pile par une règle $R_2 \neq R_1$.

Exemple

R_1 : $\mathit{Instr} \rightarrow \textbf{nom}(\mathit{Expr}, \mathit{Expr})$	État de l'automate :	Entrée :
R_2 : $\mathit{Expr} \rightarrow \textbf{nom}(\mathit{Expr}, \mathit{Expr})$	$(\mathcal{C}, \$ \ldots \textbf{nom}(\mathit{Expr}, \mathit{Expr}))$	$\ldots \$$
$\ldots$		

Réduire par R_1 (appel de fonction) ou réduire par R_2 (accès à un tableau) ?

Analyse SLR

- On veut constuire un automate **déterministe** basé sur la méthode décalage - réduction ;

Analyse SLR

- On veut constuire un automate **déterministe** basé sur la méthode décalage - réduction ;
- On va constuire un automate **fini** supplémentaire qui nous donnera l'ensemble des préfixes possibles vers lesquels les lettres déjà lues peuvent être réduites.

Analyse SLR

- On veut constuire un automate **déterministe** basé sur la méthode décalage - réduction ;
- On va constuire un automate **fini** supplémentaire qui nous donnera l'ensemble des préfixes possibles vers lesquels les lettres déjà lues peuvent être réduites.
- On gardera trace de l'**état** de cet automate grâce à la pile de l'analyseur.

Analyse SLR

- On veut constuire un automate **déterministe** basé sur la méthode décalage - réduction ;
- On va constuire un automate **fini** supplémentaire qui nous donnera l'ensemble des préfixes possibles vers lesquels les lettres déjà lues peuvent être réduites.
- On gardera trace de l'**état** de cet automate grâce à la pile de l'analyseur.
- C'est la méthode **SLR(1)** pour *Simple LR* avec connaissance d'un *token* à l'avance.

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Exemple

Items de $A \to BCD$:

$$A \to \bullet BCD$$
$$A \to B \bullet CD$$
$$A \to BC \bullet D$$
$$A \to BCD\bullet$$

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Exemple

Items de $A \to BCD$:

$$A \to \bullet BCD$$
$$A \to B \bullet CD$$
$$A \to BC \bullet D$$
$$A \to BCD \bullet$$

- Une règle $A \to \epsilon$ n'engendre que l'item $A \to \bullet$;

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Exemple

Items de $A \to BCD$:

$$A \to \bullet BCD$$
$$A \to B \bullet CD$$
$$A \to BC \bullet D$$
$$A \to BCD\bullet$$

- Une règle $A \to \epsilon$ n'engendre que l'item $A \to \bullet$;
- La **fermeture** $FI(I)$ d'un ensemble d'items I est définie par :

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Exemple

Items de $A \to BCD$:

$$A \to \bullet BCD$$
$$A \to B \bullet CD$$
$$A \to BC \bullet D$$
$$A \to BCD \bullet$$

- Une règle $A \to \epsilon$ n'engendre que l'item $A \to \bullet$;
- La **fermeture** $FI(I)$ d'un ensemble d'items I est définie par :
  1. $I \subseteq FI(I)$;

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Exemple

Items de $A \rightarrow BCD$:

$$A \rightarrow \bullet BCD$$
$$A \rightarrow B \bullet CD$$
$$A \rightarrow BC \bullet D$$
$$A \rightarrow BCD \bullet$$

- Une règle $A \rightarrow \epsilon$ n'engendre que l'item $A \rightarrow \bullet$;
- La **fermeture** $FI(I)$ d'un ensemble d'items I est définie par :
  1. $I \subseteq FI(I)$;
  2. si $A \rightarrow U \bullet BV$ est dans $FI(I)$ et $B \rightarrow W$ est une règle de la grammaire, alors $B \rightarrow \bullet W$ est dans $FI(I)$.

Items canoniques LR(0)

- Un *item* **LR(0)** est une règle de production de la grammaire avec un point repérant une **position** dans sa partie droite ;

Exemple

Items de $A \to BCD$:

$$A \to \bullet BCD$$
$$A \to B \bullet CD$$
$$A \to BC \bullet D$$
$$A \to BCD\bullet$$

- Une règle $A \to \epsilon$ n'engendre que l'item $A \to \bullet$;
- La **fermeture** $FI(I)$ d'un ensemble d'items I est définie par :
  1. $I \subseteq FI(I)$;
  2. si $A \to U \bullet BV$ est dans $FI(I)$ et $B \to W$ est une règle de la grammaire, alors $B \to \bullet W$ est dans $FI(I)$.
  3. si $A \to U \bullet BV$ est dans $FI(I)$ et $B \to \epsilon$ est une règle de la grammaire, alors $B \to \bullet$ est dans $FI(I)$.

Automate des items LR(0)

- On ajoute une règle $S' \rightarrow S$ à la grammaire ;

Automate des items LR(0)

- On ajoute une règle $S' \rightarrow S$ à la grammaire ;
- On définit l'automate des items $\mathcal{A}_I = (\mathcal{I}, I_0, \rightarrow)$ sur l'alphabet $(\Sigma \cup N)$ par :

Automate des items LR(0)

- On ajoute une règle $S' \to S$ à la grammaire ;
- On définit l'automate des items $\mathcal{A}_I = (\mathcal{I}, I_0, \to)$ sur l'alphabet $(\Sigma \cup N)$ par :
  - $\mathcal{I}$ est l'**ensemble des ensembles d'items** engendrés par la grammaire ;

Automate des items LR(0)

- On ajoute une règle $S' \to S$ à la grammaire ;
- On définit l'automate des items $\mathcal{A}_I = (\mathcal{I}, I_0, \to)$ sur l'alphabet $(\Sigma \cup N)$ par :
  - $\mathcal{I}$ est l'**ensemble des ensembles d'items** engendrés par la grammaire ;
  - I_0 est la fermeture de $\{S' \to \bullet S\}$;

Automate des items LR(0)

- On ajoute une règle $S' \to S$ à la grammaire ;
- On définit l'automate des items $\mathcal{A}_I = (\mathcal{I}, I_0, \to)$ sur l'alphabet $(\Sigma \cup N)$ par :
  - $\mathcal{I}$ est l'**ensemble des ensembles d'items** engendrés par la grammaire ;
  - I_0 est la fermeture de $\{S' \to \bullet S\}$;
  - $I \xrightarrow{A} J$ ssi J est la fermeture de l'ensemble I' tel que $B \to U \bullet AV$ est dans I ssi $B \to UA \bullet V$ est dans I' ;

Automate des items LR(0)

- On ajoute une règle $S' \to S$ à la grammaire ;
- On définit l'automate des items $\mathcal{A}_I = (\mathcal{I}, I_0, \to)$ sur l'alphabet $(\Sigma \cup N)$ par :
  - $\mathcal{I}$ est l'**ensemble des ensembles d'items** engendrés par la grammaire ;
  - I_0 est la fermeture de $\{S' \to \bullet S\}$;
  - $I \xrightarrow{A} J$ ssi J est la fermeture de l'ensemble I' tel que $B \to U \bullet AV$ est dans I ssi $B \to UA \bullet V$ est dans I' ;
  - pas d'état accepteur (pas utile pour ce qu'on va en faire).

Automate des items LR(0)

- On ajoute une règle $S' \to S$ à la grammaire ;
- On définit l'automate des items $\mathcal{A}_I = (\mathcal{I}, I_0, \to)$ sur l'alphabet $(\Sigma \cup N)$ par :
  - $\mathcal{I}$ est l'**ensemble des ensembles d'items** engendrés par la grammaire ;
  - I_0 est la fermeture de $\{S' \to \bullet S\}$;
  - $I \xrightarrow{A} J$ ssi J est la fermeture de l'ensemble I' tel que $B \to U \bullet AV$ est dans I ssi $B \to UA \bullet V$ est dans I' ;
  - pas d'état accepteur (pas utile pour ce qu'on va en faire).
- C'est équivalent à définir les états comme **les items** et des ϵ-transitions entre les items $A \to U \bullet BV$ et $B \to \bullet W$ puis à **déterminiser**.

Automate des items LR(0) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow T|S+T \\
T &\rightarrow F|T*F \\
F &\rightarrow (S)|n
\end{aligned}
$$

$$
I_0 = \left\{ \begin{array}{l} S' \rightarrow \bullet S \\ \\ \\ \\ \\ \\ \end{array} \right.
$$

Automate des items LR(0) : Exemple

Exemple

$$S' \to S$$
$$S \to T | S + T$$
$$T \to F | T * F$$
$$F \to (S) | n$$

$$I_0 = \left\{ \begin{array}{l} S' \to \bullet S \\ S \to \bullet T \\ S \to \bullet S + T \\ \\ \\ \\ \end{array} \right.$$

Automate des items LR(0) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow T | S + T \\
T &\rightarrow F | T * F \\
F &\rightarrow (S) | n
\end{aligned}
$$

$$
I_0 = \left\{
\begin{array}{l}
S' \rightarrow \bullet S \\
S \rightarrow \bullet T \\
S \rightarrow \bullet S + T \\
T \rightarrow \bullet F \\
T \rightarrow \bullet T * F \\
\\
\\
\end{array}
\right.
$$

Automate des items LR(0) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow T | S + T \\
T &\rightarrow F | T * F \\
F &\rightarrow (S) | n
\end{aligned}
$$

$$
I_0 = \begin{cases}
S' \rightarrow \bullet S \\
S \rightarrow \bullet T \\
S \rightarrow \bullet S + T \\
T \rightarrow \bullet F \\
T \rightarrow \bullet T * F \\
F \rightarrow \bullet (S) \\
F \rightarrow \bullet n
\end{cases}
$$

Automate des items LR(0) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow T | S + T \\
T &\rightarrow F | T * F \\
F &\rightarrow (S) | n
\end{aligned}
$$

$$
I_0 = \begin{cases}
S' \rightarrow \bullet S \\
S \rightarrow \bullet T \\
S \rightarrow \bullet S + T \\
T \rightarrow \bullet F \\
T \rightarrow \bullet T * F \\
F \rightarrow \bullet (S) \\
F \rightarrow \bullet n
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

Automate des items LR(0) : Exemple

Exemple

$$\begin{aligned} S' &\to S \\ S &\to T | S + T \\ T &\to F | T * F \\ F &\to (S) | n \end{aligned}$$

$$I_0 = \begin{cases} S' \to \bullet S \\ S \to \bullet T \\ S \to \bullet S + T \\ T \to \bullet F \\ T \to \bullet T * F \\ F \to \bullet (S) \\ F \to \bullet n \end{cases}$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$I_1 = \begin{cases} S' \to S \bullet \\ S \to S \bullet + T \end{cases}$$

Automate des items LR(0) : Exemple

Exemple

$$
\begin{aligned}
S' &\to S \\
S &\to T | S + T \\
T &\to F | T * F \\
F &\to (S) | n
\end{aligned}
$$

$$
I_0 = \begin{cases}
S' \to \bullet S \\
S \to \bullet T \\
S \to \bullet S + T \\
T \to \bullet F \\
T \to \bullet T * F \\
F \to \bullet (S) \\
F \to \bullet n
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$
I_1 = \begin{cases}
S' \to S \bullet \\
S \to S \bullet + T
\end{cases}
$$

$$
I_2 = \begin{cases}
S \to T \bullet \\
T \to T \bullet * F
\end{cases}
$$

Automate des items LR(0) : Exemple

Exemple

$$S' \to S$$
$$S \to T | S + T$$
$$T \to F | T * F$$
$$F \to (S) | n$$

$$I_0 = \begin{cases} S' \to \bullet S \\ S \to \bullet T \\ S \to \bullet S + T \\ T \to \bullet F \\ T \to \bullet T * F \\ F \to \bullet (S) \\ F \to \bullet n \end{cases}$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$I_1 = \begin{cases} S' \to S\bullet \\ S \to S \bullet + T \end{cases}$$

$$I_3 = \{ \; T \to F\bullet$$

$$I_2 = \begin{cases} S \to T\bullet \\ T \to T \bullet * F \end{cases}$$

Automate des items LR(0) : Exemple

Exemple

$$\begin{array}{l} S' \rightarrow S \\ S \rightarrow T | S + T \\ T \rightarrow F | T * F \\ F \rightarrow (S) | n \end{array}$$

$$I_0 = \left\{ \begin{array}{l} S' \rightarrow \bullet S \\ S \rightarrow \bullet T \\ S \rightarrow \bullet S + T \\ T \rightarrow \bullet F \\ T \rightarrow \bullet T * F \\ F \rightarrow \bullet (S) \\ F \rightarrow \bullet n \end{array} \right.$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$I_1 = \left\{ \begin{array}{l} S' \rightarrow S \bullet \\ S \rightarrow S \bullet + T \end{array} \right.$$

$$I_3 = \left\{ \ T \rightarrow F \bullet \right.$$

$$I_2 = \left\{ \begin{array}{l} S \rightarrow T \bullet \\ T \rightarrow T \bullet * F \end{array} \right.$$

$$I_5 = \left\{ \ F \rightarrow n \bullet \right.$$

Automate des items LR(0) : Exemple

Exemple

$$S' \to S$$
$$S \to T | S + T$$
$$T \to F | T * F$$
$$F \to (S) | n$$

$$I_0 = \begin{cases} S' \to \bullet S \\ S \to \bullet T \\ S \to \bullet S + T \\ T \to \bullet F \\ T \to \bullet T * F \\ F \to \bullet (S) \\ F \to \bullet n \end{cases}$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$I_1 = \begin{cases} S' \to S \bullet \\ S \to S \bullet + T \end{cases}$$

$$I_2 = \begin{cases} S \to T \bullet \\ T \to T \bullet * F \end{cases}$$

$$I_3 = \{ \; T \to F \bullet$$

$$I_5 = \{ \; F \to n \bullet$$

$$I_4 = \begin{cases} \\ \\ \end{cases}$$

Automate des items LR(0) : Exemple

Exemple

$$
\begin{array}{l}
S' \to S \\
S \to T | S + T \\
T \to F | T * F \\
F \to (S) | n
\end{array}
$$

$$
I_0 = \begin{cases}
S' \to \bullet S \\
S \to \bullet T \\
S \to \bullet S + T \\
T \to \bullet F \\
T \to \bullet T * F \\
F \to \bullet (S) \\
F \to \bullet n
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$
I_1 = \begin{cases}
S' \to S \bullet \\
S \to S \bullet + T
\end{cases}
$$

$$
I_2 = \begin{cases}
S \to T \bullet \\
T \to T \bullet * F
\end{cases}
$$

$$
I_3 = \{ \; T \to F \bullet
$$

$$
I_5 = \{ \; F \to n \bullet
$$

$$
I_4 = \begin{cases}
F \to (\bullet S) \\
\\
\\
\end{cases}
$$

Automate des items LR(0) : Exemple

Exemple

$$S' \to S$$
$$S \to T | S + T$$
$$T \to F | T * F$$
$$F \to (S) | n$$

$$I_0 = \begin{cases} S' \to \bullet S \\ S \to \bullet T \\ S \to \bullet S + T \\ T \to \bullet F \\ T \to \bullet T * F \\ F \to \bullet (S) \\ F \to \bullet n \end{cases}$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$I_1 = \begin{cases} S' \to S \bullet \\ S \to S \bullet + T \end{cases}$$

$$I_2 = \begin{cases} S \to T \bullet \\ T \to T \bullet * F \end{cases}$$

$$I_3 = \{ \ T \to F \bullet$$

$$I_5 = \{ \ F \to n \bullet$$

$$I_4 = \begin{cases} F \to (\bullet S) \\ S \to \bullet T \\ S \to \bullet S + T \\ \\ \\ \\ \end{cases}$$

Automate des items LR(0) : Exemple

Exemple

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow T | S + T \\ T &\rightarrow F | T * F \\ F &\rightarrow (S) | n \end{aligned}$$

$$I_0 = \begin{cases} S' \rightarrow \bullet S \\ S \rightarrow \bullet T \\ S \rightarrow \bullet S + T \\ T \rightarrow \bullet F \\ T \rightarrow \bullet T * F \\ F \rightarrow \bullet (S) \\ F \rightarrow \bullet n \end{cases}$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$I_1 = \begin{cases} S' \rightarrow S \bullet \\ S \rightarrow S \bullet + T \end{cases}$$

$$I_2 = \begin{cases} S \rightarrow T \bullet \\ T \rightarrow T \bullet * F \end{cases}$$

$$I_3 = \{ \; T \rightarrow F \bullet$$

$$I_5 = \{ \; F \rightarrow n \bullet$$

$$I_4 = \begin{cases} F \rightarrow (\bullet S) \\ S \rightarrow \bullet T \\ S \rightarrow \bullet S + T \\ T \rightarrow \bullet F \\ T \rightarrow \bullet T * F \end{cases}$$

Automate des items LR(0) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow T|S+T \\
T &\rightarrow F|T*F \\
F &\rightarrow (S)|n
\end{aligned}
$$

$$
I_0 = \left\{
\begin{array}{l}
S' \rightarrow \bullet S \\
S \rightarrow \bullet T \\
S \rightarrow \bullet S + T \\
T \rightarrow \bullet F \\
T \rightarrow \bullet T * F \\
F \rightarrow \bullet (S) \\
F \rightarrow \bullet n
\end{array}
\right.
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{T} I_2$

$I_0 \xrightarrow{F} I_3$

$I_0 \xrightarrow{(} I_4$

$I_0 \xrightarrow{n} I_5$

$$
I_1 = \left\{
\begin{array}{l}
S' \rightarrow S\bullet \\
S \rightarrow S \bullet + T
\end{array}
\right.
$$

$$
I_2 = \left\{
\begin{array}{l}
S \rightarrow T\bullet \\
T \rightarrow T \bullet * F
\end{array}
\right.
$$

$$
I_3 = \left\{ \; T \rightarrow F\bullet \right.
$$

$$
I_5 = \left\{ \; F \rightarrow n\bullet \right.
$$

$$
I_4 = \left\{
\begin{array}{l}
F \rightarrow (\bullet S) \\
S \rightarrow \bullet T \\
S \rightarrow \bullet S + T \\
T \rightarrow \bullet F \\
T \rightarrow \bullet T * F \\
F \rightarrow \bullet (S) \\
F \rightarrow \bullet n
\end{array}
\right.
$$

Automate des items LR(0) : Exercice

Exercice

Calculez l'automate des items LR(0) pour la grammaire suivante :

$$\begin{aligned} S &\rightarrow ACcaB \\ A &\rightarrow aB|\epsilon \\ B &\rightarrow Bb|c \\ C &\rightarrow bAb|a \end{aligned}$$

Analyseur SLR(1)

- On va utiliser la pile de l'analyseur pour mémoriser l'**exécution courante** de l'automate des items ;

Analyseur SLR(1)

- On va utiliser la pile de l'analyseur pour mémoriser l'**exécution courante** de l'automate des items ;
- Une telle exécution a la forme :

$$I_0 \xrightarrow{a_0} I_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} I_n$$

Analyseur SLR(1)

- On va utiliser la pile de l'analyseur pour mémoriser l'**exécution courante** de l'automate des items ;
- Une telle exécution a la forme :

$$I_0 \xrightarrow{a_0} I_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} I_n$$

- Comme l'automate des items est **déterministe**, il suffit de mémoriser la suite des **états** :

$$I_0, I_1, \dots, I_n$$

Analyseur SLR(1)

- On va utiliser la pile de l'analyseur pour mémoriser l'**exécution courante** de l'automate des items ;
- Une telle exécution a la forme :

$$I_0 \xrightarrow{a_0} I_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} I_n$$

- Comme l'automate des items est **déterministe**, il suffit de mémoriser la suite des **états** :

$$I_0, I_1, \ldots, I_n$$

- Si I_n est sur le dessus de la pile,

Analyseur SLR(1)

- On va utiliser la pile de l'analyseur pour mémoriser l'**exécution courante** de l'automate des items ;
- Une telle exécution a la forme :

$$I_0 \xrightarrow{a_0} I_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} I_n$$

- Comme l'automate des items est **déterministe**, il suffit de mémoriser la suite des **états** :

$$I_0, I_1, \ldots, I_n$$

- Si I_n est sur le dessus de la pile,
  - **décaler** $a \in \Sigma$, c'est **empiler** I_{n+1} tel que $I_n \xrightarrow{a} I_{n+1}$;

Analyseur SLR(1)

- On va utiliser la pile de l'analyseur pour mémoriser l'**exécution courante** de l'automate des items ;
- Une telle exécution a la forme :

$$I_0 \xrightarrow{a_0} I_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} I_n$$

- Comme l'automate des items est **déterministe**, il suffit de mémoriser la suite des **états** :

$$I_0, I_1, \dots, I_n$$

- Si I_n est sur le dessus de la pile,
  - **décaler** $a \in \Sigma$, c'est **empiler** I_{n+1} tel que $I_n \xrightarrow{a} I_{n+1}$;
  - **réduire** par $A \to B_1 \dots B_k$ c'est **dépiler** les k derniers états et empiler I' tel que $I_{n-k-1} \xrightarrow{A} I'$.

Analyseur SLR(1) : automate à pile

- États : $\mathcal{I}$ (initial), $\mathcal{F}$ (accepteur) et $\mathcal{C}$

Analyseur SLR(1) : automate à pile

- États : $\mathcal{I}$ (initial), $\mathcal{F}$ (accepteur) et $\mathcal{C}$
- Pour toute règle $A \to B_1 \dots B_k$, $A \neq S'$, pour tout $a \in \text{SUIVANT}(A)$ et tout état I de $\mathcal{A}_I$ tel que $A \to B_1 \dots B_k\bullet \in I$, si I' est le $k+1$-ème état dans la pile et $I' \xrightarrow{A} I''$,
  En étendant l'automate pour dépiler k symboles quelconques en une transition
  $$\delta(\mathcal{C}, \epsilon(a), *^{\mathbf{k}}) = \{(\mathcal{C}, \mathbf{I''})\}$$

Analyseur SLR(1) : automate à pile

- États : $\mathcal{I}$ (initial), $\mathcal{F}$ (accepteur) et $\mathcal{C}$
- Pour toute règle $A \to B_1 \dots B_k$, $A \neq S'$, pour tout $a \in \text{SUIVANT}(A)$ et tout état I de $\mathcal{A}_I$ tel que $A \to B_1 \dots B_k \bullet \in I$, si I' est le $k+1$-ème état dans la pile et $I' \xrightarrow{A} I''$,
  En étendant l'automate pour dépiler k symboles quelconques en une transition
  $$\delta(\mathcal{C}, \epsilon(a), *^{\mathbf{k}}) = \{(\mathcal{C}, \mathbf{I''})\}$$
- Pour tout terminal a et tous les états I, I' de $\mathcal{A}_I$ tels que $I \xrightarrow{a} I'$,
  $$\delta(\mathcal{C}, a, \mathbf{I}) = \{(\mathcal{C}, \mathbf{II'})\}$$

Analyseur SLR(1) : automate à pile

- États : $\mathcal{I}$ (initial), $\mathcal{F}$ (accepteur) et $\mathcal{C}$
- Pour toute règle $A \to B_1 \dots B_k$, $A \neq S'$, pour tout $a \in$ SUIVANT(A) et tout état I de $\mathcal{A}_I$ tel que $A \to B_1 \dots B_k \bullet \in I$, si I' est le $k+1$-ème état dans la pile et $I' \xrightarrow{A} I''$,
  En étendant l'automate pour dépiler k symboles quelconques en une transition
  $$\delta(\mathcal{C}, \epsilon(a), *^{\mathbf{k}}) = \{(\mathcal{C}, \mathbf{I''})\}$$
- Pour tout terminal a et tous les états I, I' de $\mathcal{A}_I$ tels que $I \xrightarrow{a} I'$,
  $$\delta(\mathcal{C}, a, \mathbf{I}) = \{(\mathcal{C}, \mathbf{II'})\}$$
- Initialisation :
  $$\delta(\mathcal{I}, \epsilon(\Sigma), \epsilon) = \{(\mathcal{C}, \mathbf{I_0})\}$$

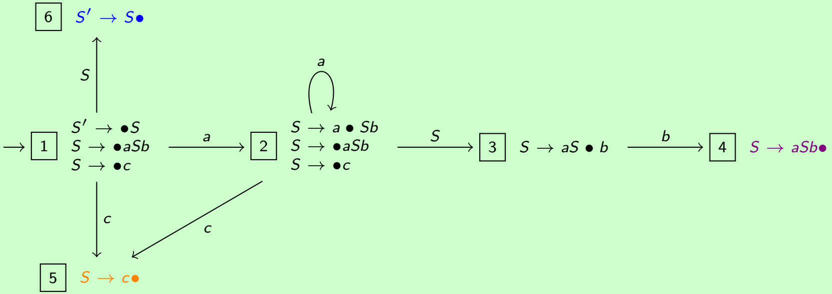
Analyseur SLR(1) : automate à pile

- États : $\mathcal{I}$ (initial), $\mathcal{F}$ (accepteur) et $\mathcal{C}$
- Pour toute règle $A \to B_1 \dots B_k$, $A \neq S'$, pour tout $a \in \text{SUIVANT}(A)$ et tout état I de $\mathcal{A}_I$ tel que $A \to B_1 \dots B_k \bullet \in I$, si I' est le $k+1$-ème état dans la pile et $I' \xrightarrow{A} I''$,
  En étendant l'automate pour dépiler k symboles quelconques en une transition
  $$\delta(\mathcal{C}, \epsilon(a), *^{\mathbf{k}}) = \{(\mathcal{C}, \mathbf{I''})\}$$
- Pour tout terminal a et tous les états I, I' de $\mathcal{A}_I$ tels que $I \xrightarrow{a} I'$,
  $$\delta(\mathcal{C}, a, \mathbf{I}) = \{(\mathcal{C}, \mathbf{II'})\}$$
- Initialisation :
  $$\delta(\mathcal{I}, \epsilon(\Sigma), \epsilon) = \{(\mathcal{C}, \mathbf{I_0})\}$$
- Acceptation : pour tout I tel que $S' \to S \bullet \in I$:
  $$\delta(\mathcal{C}, \$, I) = \{(\mathcal{F}, \epsilon)\}$$

Analyseur SLR(1)

Exemple

$G = \{S \to aSb | c\}$ $\qquad$ $\text{SUIVANT}(S) = \{\$, b\}$



Exemple d'**exécution** de l'automate **à pile** sur *aaacbbb*\$:

$$
\begin{aligned}
(\mathcal{I}, \epsilon) &\xrightarrow{\epsilon(\Sigma)} (\mathcal{C}, 1) \xrightarrow{a} (\mathcal{C}, 12) \xrightarrow{a} (\mathcal{C}, 122) \xrightarrow{a} (\mathcal{C}, 1222) \xrightarrow{c} (\mathcal{C}, 12225) \\
&\xrightarrow{\epsilon(b)} (\mathcal{C}, 12223) \xrightarrow{b} (\mathcal{C}, 122234) \xrightarrow{\epsilon(b)} (\mathcal{C}, 1223) \xrightarrow{b} (\mathcal{C}, 12234) \\
&\xrightarrow{\epsilon(b)} (\mathcal{C}, 123) \xrightarrow{b} (\mathcal{C}, 1234) \xrightarrow{\epsilon(\$)} (\mathcal{C}, 16) \xrightarrow{\$} (\mathcal{F}, \epsilon)
\end{aligned}
$$

Analyseur SLR(1)

- Si l'automate à pile obtenu est **déterministe** alors l'analyse SLR a réussi ;

Analyseur SLR(1)

- Si l'automate à pile obtenu est **déterministe** alors l'analyse SLR a réussi ;
- On dit alors que la grammaire est **SLR(1)** ;

Analyseur SLR(1)

- Si l'automate à pile obtenu est **déterministe** alors l'analyse SLR a réussi ;
- On dit alors que la grammaire est **SLR(1)** ;
- Sinon, il faut une technique plus puissante : l'analyse **LR**.

Analyseur LR(1)

- Dans l'analyse SLR, on réduit par $A \rightarrow W$ quand :

Analyseur LR(1)

- Dans l'analyse SLR, on réduit par $A \to W$ quand :
  1. $A \to W\bullet$ est dans l'état au sommet de la pile ;

Analyseur LR(1)

- Dans l'analyse SLR, on réduit par $A \to W$ quand :
  1. $A \to W\bullet$ est dans l'état au sommet de la pile ;
  2. et la prochaine lettre a est dans SUIVANT(A).

Analyseur LR(1)

- Dans l'analyse SLR, on réduit par $A \to W$ quand :
  1. $A \to W\bullet$ est dans l'état au sommet de la pile ;
  2. et la prochaine lettre a est dans SUIVANT(A).
- **Mais** rien ne dit que pour le préfixe actuel, A puisse **vraiment** être suivi de a ;

Analyseur LR(1)

- Dans l'analyse SLR, on réduit par $A \to W$ quand :
    1. $A \to W\bullet$ est dans l'état au sommet de la pile ;
    2. et la prochaine lettre a est dans SUIVANT(A).
- **Mais** rien ne dit que pour le préfixe actuel, A puisse **vraiment** être suivi de a ;
- On va **raffiner** la notion d'item.

Items canoniques LR(1)

- Un **item LR(1)** est un item plus un terminal : $(A \to U \bullet W, a)$;

Items canoniques LR(1)

- Un **item LR(1)** est un item plus un terminal : $(A \to U \bullet W, a)$;
- On redéfinit la **fermeture** $FI(I)$ pour un ensemble I d'items $LR(1)$:

Items canoniques LR(1)

- Un **item LR(1)** est un item plus un terminal : $(A \rightarrow U \bullet W, a)$;
- On redéfinit la **fermeture** $FI(I)$ pour un ensemble I d'items $LR(1)$:
  1. $I \subseteq FI(I)$;

Items canoniques LR(1)

- Un **item LR(1)** est un item plus un terminal : $(A \rightarrow U \bullet W, a)$;
- On redéfinit la **fermeture** $FI(I)$ pour un ensemble I d'items $LR(1)$:
  1. $I \subseteq FI(I)$;
  2. si $(A \rightarrow U \bullet BV, a)$ est dans $FI(I)$ et $B \rightarrow W$ est une règle de la grammaire, alors pour tout terminal b **dans** PREMIER(Va), $(B \rightarrow \bullet W, b)$ est dans $FI(I)$.

Items canoniques LR(1)

- Un **item LR(1)** est un item plus un terminal : $(A \to U \bullet W, a)$;
- On redéfinit la **fermeture** $FI(I)$ pour un ensemble I d'items $LR(1)$:
  1. $I \subseteq FI(I)$;
  2. si $(A \to U \bullet BV, a)$ est dans $FI(I)$ et $B \to W$ est une règle de la grammaire, alors pour tout terminal b **dans** PREMIER(Va), $(B \to \bullet W, b)$ est dans $FI(I)$.
- On définit alors l'**automate des items LR(1)** comme précédemment avec l'**état initial** :

$$I_0 = FI(\{(S' \to \bullet S, \$)\})$$

Automate des items LR(1) : Exemple

Exemple

$$S' \to S$$
$$S \to CC$$
$$C \to cC|d$$

$$I_0 = \left\{ \begin{array}{l} (S' \to \bullet S, \$) \\ \\ \\ \\ \end{array} \right.$$

Automate des items LR(1) : Exemple

Exemple

$$S' \to S$$
$$S \to CC$$
$$C \to cC | d$$

$$I_0 = \begin{cases} (S' \to \bullet S, \$) \\ (S \to \bullet CC, \$) \\ \\ \\ \\ \end{cases}$$

Automate des items LR(1) : Exemple

Exemple

$$S' \to S$$
$$S \to CC$$
$$C \to cC|d$$

$$I_0 = \left\{ \begin{array}{l} (S' \to \bullet S, \$) \\ (S \to \bullet CC, \$) \\ (C \to \bullet cC, c) \\ (C \to \bullet cC, d) \\ \\ \\ \end{array} \right.$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow CC \\
C &\rightarrow cC | d
\end{aligned}
$$

$$
I_0 = \begin{cases}
(S' \rightarrow \bullet S, \$) \\
(S \rightarrow \bullet CC, \$) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

Automate des items LR(1) : Exemple

Exemple

$$S' \to S$$
$$S \to CC$$
$$C \to cC|d$$

$$I_0 = \begin{cases} (S' \to \bullet S, \$) \\ (S \to \bullet CC, \$) \\ (C \to \bullet cC, c) \\ (C \to \bullet cC, d) \\ (C \to \bullet d, c) \\ (C \to \bullet d, d) \end{cases}$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\to S \\
S &\to CC \\
C &\to cC|d
\end{aligned}
$$

$$
I_0 = \begin{cases}
(S' \to \bullet S, \$) \\
(S \to \bullet CC, \$) \\
(C \to \bullet cC, c) \\
(C \to \bullet cC, d) \\
(C \to \bullet d, c) \\
(C \to \bullet d, d)
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$I_1 = \{\ (S' \to S\bullet, \$)$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\to S \\
S &\to CC \\
C &\to cC|d
\end{aligned}
$$

$$
I_0 = \begin{cases}
(S' \to \bullet S, \$) \\
(S \to \bullet CC, \$) \\
(C \to \bullet cC, c) \\
(C \to \bullet cC, d) \\
(C \to \bullet d, c) \\
(C \to \bullet d, d)
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$I_1 = \{ \ (S' \to S\bullet, \$)$$

$$
I_2 = \begin{cases}
(S \to C \bullet C, \$) \\
\\
\end{cases}
$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow CC \\
C &\rightarrow cC|d
\end{aligned}
$$

$$
I_0 = \begin{cases}
(S' \rightarrow \bullet S, \$) \\
(S \rightarrow \bullet CC, \$) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$
I_1 = \{\ (S' \rightarrow S\bullet, \$)
$$

$$
I_2 = \begin{cases}
(S \rightarrow C \bullet C, \$) \\
(C \rightarrow \bullet cC, \$) \\
(C \rightarrow \bullet d, \$)
\end{cases}
$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow CC \\
C &\rightarrow cC|d
\end{aligned}
$$

$$
I_0 = \left\{ \begin{array}{l} (S' \rightarrow \bullet S, \$) \\ (S \rightarrow \bullet CC, \$) \\ (C \rightarrow \bullet cC, c) \\ (C \rightarrow \bullet cC, d) \\ (C \rightarrow \bullet d, c) \\ (C \rightarrow \bullet d, d) \end{array} \right.
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$
I_1 = \{ \ (S' \rightarrow S\bullet, \$)
$$

$$
I_2 = \left\{ \begin{array}{l} (S \rightarrow C \bullet C, \$) \\ (C \rightarrow \bullet cC, \$) \\ (C \rightarrow \bullet d, \$) \end{array} \right.
$$

$$
I_3 = \left\{ \begin{array}{l} (C \rightarrow c \bullet C, c) \\ (C \rightarrow c \bullet C, d) \end{array} \right.
$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow CC \\
C &\rightarrow cC|d
\end{aligned}
$$

$$
I_0 = \begin{cases}
(S' \rightarrow \bullet S, \$) \\
(S \rightarrow \bullet CC, \$) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$
I_1 = \{ \; (S' \rightarrow S\bullet, \$)
$$

$$
I_2 = \begin{cases}
(S \rightarrow C \bullet C, \$) \\
(C \rightarrow \bullet cC, \$) \\
(C \rightarrow \bullet d, \$)
\end{cases}
$$

$$
I_3 = \begin{cases}
(C \rightarrow c \bullet C, c) \\
(C \rightarrow c \bullet C, d) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet d, c) \\
\\
\end{cases}
$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{array}{l}
S' \rightarrow S \\
S \rightarrow CC \\
C \rightarrow cC|d
\end{array}
$$

$$
I_0 = \begin{cases}
(S' \rightarrow \bullet S, \$) \\
(S \rightarrow \bullet CC, \$) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$
I_1 = \{ \ (S' \rightarrow S\bullet, \$)
$$

$$
I_2 = \begin{cases}
(S \rightarrow C \bullet C, \$) \\
(C \rightarrow \bullet cC, \$) \\
(C \rightarrow \bullet d, \$)
\end{cases}
$$

$$
I_3 = \begin{cases}
(C \rightarrow c \bullet C, c) \\
(C \rightarrow c \bullet C, d) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

Automate des items LR(1) : Exemple

Exemple

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow CC \\
C &\rightarrow cC|d
\end{aligned}
$$

$$
I_0 = \begin{cases}
(S' \rightarrow \bullet S, \$) \\
(S \rightarrow \bullet CC, \$) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

Transitions possibles :

$I_0 \xrightarrow{S} I_1$

$I_0 \xrightarrow{C} I_2$

$I_0 \xrightarrow{c} I_3$

$I_0 \xrightarrow{d} I_4$

$$
I_1 = \{ \ (S' \rightarrow S \bullet, \$)
$$

$$
I_2 = \begin{cases}
(S \rightarrow C \bullet C, \$) \\
(C \rightarrow \bullet cC, \$) \\
(C \rightarrow \bullet d, \$)
\end{cases}
$$

$$
I_3 = \begin{cases}
(C \rightarrow c \bullet C, c) \\
(C \rightarrow c \bullet C, d) \\
(C \rightarrow \bullet cC, c) \\
(C \rightarrow \bullet d, c) \\
(C \rightarrow \bullet cC, d) \\
(C \rightarrow \bullet d, d)
\end{cases}
$$

$$
I_4 = \begin{cases}
(C \rightarrow d \bullet, c) \\
(C \rightarrow d \bullet, d)
\end{cases}
$$

$\dots$

Automate des items LR(1) : Exercice

Exercice

Calculez l'automate des items LR(1) pour la grammaire suivante :

$$
\begin{aligned}
S &\rightarrow ACB \\
A &\rightarrow aA|b \\
B &\rightarrow Bb|a \\
C &\rightarrow aBcAb
\end{aligned}
$$

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \to W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \to W\bullet, a)$ est dans l'état du sommet de la pile ;

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \to W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \to W\bullet, a)$ est dans l'état du sommet de la pile ;
- Si l'automate obtenu est **déterministe** alors on dit que la grammaire est LR(1) ;

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \to W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \to W\bullet, a)$ est dans l'état du sommet de la pile ;
- Si l'automate obtenu est **déterministe** alors on dit que la grammaire est LR(1) ;
- Sinon, on peut essayer de résoudre les conflits en introduisant dans la grammaire :

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \rightarrow W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \rightarrow W\bullet, a)$ est dans l'état du sommet de la pile ;
- Si l'automate obtenu est **déterministe** alors on dit que la grammaire est LR(1) ;
- Sinon, on peut essayer de résoudre les conflits en introduisant dans la grammaire :
  - des **priorités** ;

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \to W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \to W\bullet, a)$ est dans l'état du sommet de la pile ;
- Si l'automate obtenu est **déterministe** alors on dit que la grammaire est LR(1) ;
- Sinon, on peut essayer de résoudre les conflits en introduisant dans la grammaire :
  - des **priorités** ;
  - et - ou - de l'**associativité** ;

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \to W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \to W\bullet, a)$ est dans l'état du sommet de la pile ;
- Si l'automate obtenu est **déterministe** alors on dit que la grammaire est LR(1) ;
- Sinon, on peut essayer de résoudre les conflits en introduisant dans la grammaire :
  - des **priorités** ;
  - et - ou - de l'**associativité** ;
- L'automate des items et l'analyseur LR(1) sont considérablement **plus gros** que dans le cas SLR ;

Analyseur LR(1)

- L'analyseur LR(1) est fait avec **presque** le même automate à pile que l'analyseur SLR(1) ;
- La **seule** différence est qu'on ne réduit par $A \rightarrow W$ que, si la prochaine lettre du mot à lire est a, lorsque l'item LR(1) $(A \rightarrow W\bullet, a)$ est dans l'état du sommet de la pile ;
- Si l'automate obtenu est **déterministe** alors on dit que la grammaire est LR(1) ;
- Sinon, on peut essayer de résoudre les conflits en introduisant dans la grammaire :
    - des **priorités** ;
    - et - ou - de l'**associativité** ;
- L'automate des items et l'analyseur LR(1) sont considérablement **plus gros** que dans le cas SLR ;
- On peut **fusionner** les items LR(1) qui partagent la même partie LR(0) pour réduire cette taille : c'est l'analyse LALR (*Lookahead LR*). Bon compromis entre LR et SLR

Plan

Un constructeur d'analyseur syntaxique : Yacc

- **YACC** est un constructeur d'analyseur syntaxique pour Unix ;

Un constructeur d'analyseur syntaxique : Yacc

- **YACC** est un constructeur d'analyseur syntaxique pour Unix ;
- Il génère un **analyseur LALR** ;

Un constructeur d'analyseur syntaxique : Yacc

- **YACC** est un constructeur d'analyseur syntaxique pour Unix ;
- Il génère un **analyseur LALR** ;
- Un outil ayant ses fonctionalités fait partie du standard **POSIX** ;

Un constructeur d'analyseur syntaxique : Yacc

- **YACC** est un constructeur d'analyseur syntaxique pour Unix ;
- Il génère un **analyseur LALR** ;
- Un outil ayant ses fonctionalités fait partie du standard **POSIX** ;
- En pratique, on en utilise souvent une implémentation du projet GNU : **Bison**.

Un constructeur d'analyseur syntaxique : Yacc

- **YACC** est un constructeur d'analyseur syntaxique pour Unix ;
- Il génère un **analyseur LALR** ;
- Un outil ayant ses fonctionalités fait partie du standard **POSIX** ;
- En pratique, on en utilise souvent une implémentation du projet GNU : **Bison**.

Syntaxe du fichier de définition :

```
Définitions
%%
Règles de la grammaire
%%
Code C
```

Yacc – Exemple : eval.y

```
%{
#include "..."
%}
%token          nombre
%start          EXPRCALCS
%%
EXPRCALCS :     EXPRCALC
                | EXPRCALCS EXPRCALC
                ;
EXPRCALC :      EXPR '='              {printf ("%d\ n", $1);
                ;
EXPR :          FACTEUR
                | EXPR '+' FACTEUR    {$$ = $1 + $3;}
                ;
FACTEUR :       nombre                {$$ = $1;}
                ;
%%...
```

Plan

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;
- On peut générer **automatiquement** des analyseurs pour des sous-ensembles **suffisamment expressifs** de ces grammaires ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;
- On peut générer **automatiquement** des analyseurs pour des sous-ensembles **suffisamment expressifs** de ces grammaires ;
- Ces analyseurs sont des **automates à pile** déterministes ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;
- On peut générer **automatiquement** des analyseurs pour des sous-ensembles **suffisamment expressifs** de ces grammaires ;
- Ces analyseurs sont des **automates à pile** déterministes ;
- Les automates à piles s'implémentent facilement et systématiquement ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;
- On peut générer **automatiquement** des analyseurs pour des sous-ensembles **suffisamment expressifs** de ces grammaires ;
- Ces analyseurs sont des **automates à pile** déterministes ;
- Les automates à piles s'implémentent facilement et systématiquement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;
- On peut générer **automatiquement** des analyseurs pour des sous-ensembles **suffisamment expressifs** de ces grammaires ;
- Ces analyseurs sont des **automates à pile** déterministes ;
- Les automates à piles s'implémentent facilement et systématiquement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Un certain nombre de **contraintes** ne sont **pas** exprimables avec les grammaires hors contexte ;

Conclusion

- Pour l'analyse **syntaxique**, il faut un formalisme plus puissant que les **expressions régulières** ;
- Les **grammaires hors contexte** sont un outil adapté à cette tâche ;
- On peut générer **automatiquement** des analyseurs pour des sous-ensembles **suffisamment expressifs** de ces grammaires ;
- Ces analyseurs sont des **automates à pile** déterministes ;
- Les automates à piles s'implémentent facilement et systématiquement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Un certain nombre de **contraintes** ne sont **pas** exprimables avec les grammaires hors contexte ;
- On prendra en compte ces contraintes via la phase d'**analyse sémantique** notamment.