NF11 - GRAMMAIRES

LL, LR, SLR, LALR et AntLR

1. Remarques préliminaires

- LR(0) : théoriquement importante mais trop faible
- SLR(1) : version renforcée de LR(0) mais assez faible
- LR(1) : très puissante mais nécessite une mémoire importante
- LALR(1) : version affaiblie de LR(1) puissante et exécutable

2. Grammaire LL(1)
    a. Définitions

- Une grammaire est dite LL(1) lorsque sa table d'analyse ne comporte pas plusieurs règles de production pour une même case ;
- Une grammaire ambiguë, récursive à gauche ou non factorisée à gauche n'est pas LL(1) ;
- Lorsqu'une grammaire n'est pas LL(1), il n'existe qu'une alternative : essayer de la rendre LL(1) ou choisir un analyseur syntaxique de nature différente.

    b. Premiers

Initialisation :
- ∀ a ∈ T, PREM(a) = {a}.
- ∀ A ∈ V, PREM(A) = ∅.
- Pour tout choix ou fin de choix α non vide, PREM(α) = ∅.
- PREM(ε) = ε.

Algorithme de construction des PREM(α) :
- Pour chaque règle de production A → α, ajouter les éléments de PREM(α) à ceux de PREM(A) y compris ε.
- Pour chaque choix ou fin de choix de la forme Xβ, ajouter les éléments de PREM(X) à ceux de PREM(Xβ), sauf ε.
    → Lorsque PREM(X) contient ε, ajouter les éléments de PREM(β) à ceux de PREM(α) y compris ε.

Répéter les phases précédentes tant que l'un des PREM(α) est modifié.

    c. Suivants

Initialisation :
- Pour toute variable A, SUIV(A) = ∅.
- Ajouter un marqueur de fin de chaîne à l'axiome ($ par exemple).
- Calculer les premiers pour tous les choix et fins de choix de la grammaire.

Algorithme :
- Pour chaque règle A→αBβ, ajouter les éléments de PREM(β) à ceux de SUIV(B), sauf le cas échéant.

```
   - Pour chaque règle A → αB, ajouter les éléments de SUIV(A) aux éléments
de SUIV(B).
   - Pour chaque règle A → αBβ, lorsque ε ∈ PREM(β), ajouter les éléments de
SUIV(A) aux éléments de SUIV(B).

Répéter les phases 2 et 3 tant que l'un des SUIV(A) est modifié.
```

d. Exercice

0. Analyse du problème :

Soit G la grammaire suivante :

S → ABe
A → dB | aS | c
B → AS | b

1. On peut réécrire cette grammaire, de la forme suivante :

1. S → ABe
2. A → dB
3. A → aS
4. A → c
5. B → AS
6. B → b

2. On vérifie que la grammaire ressemble à du LL(1) :

- S commence par A et a ;
- A possède toujours un terminal en premier ;
- B possède toujours un terminal en premier.
  → non récursive à gauche
  → bien factorisée à gauche également

→ La grammaire est très probablement LL(1). Pour le vérifier, on réalise la table d'analyse.

3. Réalisation de la table d'analyse :

	PREMIERS	**SUIVANTS**
S	{a,c,d}	{$,a,b,c,d,e}
A	{a,c,d}	{a,b,c,d}
B	{b,a,c,d}	{a,b,c,d,e}
ABe	{a,c,d}	/
Be	{b,a,c,d}	
AS	{a,c,d}	

Quelques explications :

- On crée une nouvelle ligne pour chaque choix composé d'une variable ;
- On cherche tous les premiers de chaque choix ;

- On cherche ensuite les suivants en commençant avec un peu de jugeote.
    - Construction des suivants de A :
      → On retrouve A à droite des règles dans les cas suivants : ABe et AS.
      → Il suffit d'ajouter à SUIV(A) les premiers de B et les premiers de S.
      → On ajoute donc {b,a,c,d} et {a,c,d}
      → On obtient bien {a,b,c,d}.

    - Construction des suivants de B :
      → On retrouve B à droite des règles dans les cas suivants : ABe et dB_.
      → Ajoutons d'abord {e} car le terminal succède directement notre variable B.
      → Ajoutons maintenant les suivants de A car rien ne succède B dans la règle A→dB_. On ajoute donc {a,b,c,d}.
      → On obtient bien {a,b,c,d,e}.

    - Construction des suivants de S :
      → On ajoute toujours $ à l'axiome. Soit {$}.
      → On retrouve S à droite des règles dans les cas suivants : aS_ et AS_.
      → Ajoutons maintenant les suivants de A car rien ne succède S dans la règle A→aS_. On ajoute donc {a,b,c,d}.
      → Ajoutons maintenant les suivants de B car rien ne succède S dans la règle B→AS_. On ajoute donc {a,b,c,d,e}.
      → On obtient bien {$,a,b,c,d,e}.

Il suffit maintenant de retranscrire ces informations, dans ce que l'on appelle une table d'analyse.

Algorithme de construction

```
Pour chaque production A → α
      pour tout a ∈ PREM(α) et a =/= ε
          ajouter A → α dans la case M[A,a]
      si ε ∈ PREM(α), alors
      pour chaque b ∈ SUIV(A), ajouter A → α dans la case M[A,b]
```

Une case vide qui devrait être utilisée lors de l'analyse correspond à une erreur de syntaxe.

On obtient :

	a	**b**	**c**	**d**	**e**	**$**
S	1		1	1		
A	3		4	2		
B	5	6	5	5		

A partir de cette table, si une case possède deux règles de production, la grammaire n'est pas LL(1).

4. Analyse de la chaîne "adbbebe" :
Si la grammaire est LL(1), on peut analyser une chaîne.

INPUT	PILE	ACTION	SORTIE
adbbebe$	S$	Production	1
adbbebe$	ABe$	Production	3
adbbebe$	aSBe$	Match	/
dbbebe$	SBe$	Production	1
dbbebe$	ABeBe$	Production	2
dbbebe$	dBBeBe$	Match	/
bbebe$	BBeBe$	Production	6
bbebe$	bBeBe$	Match	/
bebe$	BeBe$	Production	6
bebe$	beBe$	Match	/
ebe$	eBe$	Match	/
be$	Be$	Production	6
be$	be$	Match	/
e$	e$	Match	/
$	$	Acceptation	/

5. Conflits :

- Conflit PREMIER-PREMIER :
  Par exemple, S → ABb et S → BAa donnera un conflit PP
- Conflit PREMIER-SUIVANT :
  Par exemple, A → AAb et A → ε donnera un conflit PS

6. Rendre une grammaire LL(1) :

(Grammaire propre)
Une grammaire est propre si elle ne contient aucune règle de la forme suivante : A → ε.

(Grammaire immédiatement récursive à gauche)

Une grammaire est immédiatement récursive à gauche si elle contient une règle de la forme : A → Aα. Une telle grammaire possède une règle de la forme : A → β, sinon la règle précédente ne serait pas utilisable. Pour chaque premier(β), il existe au moins deux règles applicables. La grammaire n'est pas LL(1).

(Grammaire récursive à gauche)
Une grammaire est récursive à gauche si elle contient une variable A se dérivant en Aα (A $*\Rightarrow$ Aα).
Une telle grammaire n'est pas LL(1) pour la même raison que pour la récursivité à gauche immédiate. Une grammaire peut posséder une récursivité à gauche cachée. C'est le cas lorsque : A $\Rightarrow$ αA et α $\Rightarrow$ ε.

(Grammaire non factorisée à gauche)
Une grammaire n'est pas factorisée à gauche lorsque deux choix d'une même variable commencent par le même symbole.

(Elimination de la récursivité à gauche immédiate)
On observe que A → Aα et A → β, permettent de déduire que A produit des chaînes de la forme : βα*. Pour éliminer la récursivité à gauche immédiate, il suffit d'ajouter une variable A' et de remplacer les règles telles que :

A → Aα
A → β

par les règles :

A → βA'
A' → αA'|ε

3. Grammaire LR(0)
    a. Exercice

0. Analyse du problème :

Soit G la grammaire suivante :
S → aSAB | BA
A → aA | B
B → b

Notons qu'une grammaire ne peut pas être LR(0) si elle possède un choix nullifiable (hors de son atome).
Notons également d'un item présentant un décalage et une réduction en même temps implique une grammaire non LR(0).

1. On peut réécrire cette grammaire, de la forme suivante :

0. S' → S$
1. S → aSAB
2. S → BA
3. A → aA
4. A → B
5. B → b

2. Recherche des items :

//On commence par S' et on ajoute toutes les variables pour lesquelles le point est devant.

I_0 = {S' → .S$, S → .aSAB, S → .BA, B → .b}

//On traite ensuite toutes les variables A de la forme A → ... en transition.

I_1 =$\sigma(I_0$, S) = {S' → S.$}

I_2 =$\sigma(I_0$, B) = {S → B.A, A → .aA, A → .B, B → .b}

//Puis, on traite toutes les terminaux en transition.

I_3 =$\sigma(I_0$, a) = {S → a.SAB, S → .aSAB, S→ .BA, B→ .b }

I_4 =$\sigma(I_0$, b) = {B → b.} //On dit ici que le pivot est atteint.

//On continue avec les nouveaux états générés.

I_5 =$\sigma(I_1$, $) = {S' → S$.}

I_6 =$\sigma(I_2$, A) = {S → BA.}

I_7 =$\sigma(I_2$, B) = {A → B.}

I_8 =$\sigma(I_2$, a) = {A → a.A, A → .aA, A → .B, B → .b}

I_4 =$\sigma(I_2$, b) = {B → b.}

I_9 =$\sigma(I_3$, S) = {S → aS.AB, A → .aA, A → .B, B → .b}

I_2 =$\sigma(I_3$, B) = {S → B.A, A → .aA, A → .B, B → .b}

I_3 =$\sigma(I_3$, a) = {S → a.SAB, S → .aSAB, S→ .BA, B → .b}

I_4 =$\sigma(I_3$, b) = {B→ b.}

I_{10} =$\sigma(I_8$, A) = {A → aA.}

I_7 =$\sigma(I_8$, B) = {A → B.}

I_8 =$\sigma(I_8$, a) = {A → a.A, A → .aA, A → .B, B → .b}

I_4 =$\sigma(I_8$, b) = {B → b.}

I_{11} =$\sigma(I_9$, A) = {S → aSA.B, B → .b}

I_7 =$\sigma(I_9$, B) = {A → B.}

I_8 =$\sigma(I_9$, a) = {A → a.A, A → .aA, A → .B, B → .b}

I_4 =$\sigma(I_9$, b) = {B → b.}

I_{12} =$\sigma(I_{11}$, B) = {S → aSAB.}

I_4 =$\sigma(I_{11}$, b) = {B → b.}

3. Construction de la table d'analyse :

	ACTION	a	b	$	S	A	B
0	Décalage (D)	3	4		1		2
1	Décalage (D)			5			
2	Décalage (D)	8	4			6	7

3	Décalage (D)	3	4		9		2
4	Réduction (R5)						
5	Acceptation ($)						
6	Réduction (R2)						
7	Réduction (R4)						
8	Décalage (D)	8	4			10	7
9	Décalage (D)	8	4			11	7
10	Réduction (R3)						
11	Décalage (D)		4				12
12	Réduction (R1)						

Si une réduction a lieu en même temps qu'un décalage dans cette table, alors la grammaire n'est pas LR(0). Ici, **aucun conflit** n'est à relever, la grammaire est LR(0). Nous pouvons donc analyser une chaîne à partir de la table que nous venons de créer.

4. Analyse de la chaîne "abbbba$" :

Entrée	Pile	Action	Sortie
abbbba$	0⊦	Décalage	
bbbba$	3a0⊦	Décalage	
bbba$	4b3a0⊦	Réduction (R5)	5
bbba$	2B3a0⊦	Décalage	
bba$	4b2B3a0⊦	Réduction (R5)	5
bba$	7B2B3a0⊦	Réduction (R4)	4
bba$	6A2B3a0⊦	Réduction (R2)	2
bba$	9S3a0⊦	Décalage	
ba$	4b9S3a0⊦	Réduction (R5)	5
ba$	7B9S3a0⊦	Réduction (R4)	4
ba$	11A953a0⊦	Décalage	
a$	4b11A9S3a0⊦	Réduction	5
a$	12B11A9S3a0⊦	Réduction	1
a$	1S0⊦	Décalage	

$	a1S0⊢	erreur 1(a) vide	

5. Convention de Pascal :

Il ne s'agit pas ici de modifier notre grammaire en tant que telle, mais d'appliquer une convention sur la manière dont notre grammaire est appliquée algorithmiquement parlant.
La convention de PASCAL consiste à dire que le "else" est relatif au "if" le plus proche.

4. Grammaire LR(1)
   a. Exercice

0. Analyse du problème :

Considérons la grammaire suivante :
S → AA | cAc
A → aA | b

1. On peut réécrire cette grammaire, de la forme suivante :

0. S' → S$
1. S → AA
2. S → cAc
3. A → aA
4. A → b

2. Construction des items (attention aux caractères de prévision) :

I_0 = { [S' → .S$, ε], [S → .AA, $] [S → .cAc, $] [A → .aA, a|b] [A → .b, a|b] }
I_1 = $\sigma(I_0$, S) = { [S' → S.$, ε] }
I_2 = $\sigma(I_0$, A) = { [S → A.A, $] [A → .aA, $] [A → .b, $] }
I_3 = $\sigma(I_0$, a) = { [A → a.A, a|b] [A → .aA, a|b] [A → .b, a|b] }
I_4 = $\sigma(I_0$, b) = { [A → b., a|b] }
I_5 = $\sigma(I_0$, c) = { [S → c.Ac, $] [A → .aA, c] [A → .b, c] }
I_6 = $\sigma(I_1$, $) = { [S' → S$., ε] }
I_7 = $\sigma(I_2$, A) = { [S → AA., $] }
I_8 = $\sigma(I_2$, a) = { [A → a.A, $] [A → .aA, $] [A → .b, $] }
I_9 = $\sigma(I_2$, b) = { [A → b., $] }
I_{10} = $\sigma(I_3$, A) = { [A → aA., a|b] }
I_3 = $\sigma(I_3$, a) = { [A → a.A, a|b] [A → .aA, a|b] [A → .b, a|b]}
I_4 = $\sigma(I_3$, b) = { [A → b., a|b] }
I_{11} = $\sigma(I_5$, A) = { [S → cA.c, $] }
I_{12} = $\sigma(I_5$, a) = { [A → a.A, c] [A → .aA, c] [A → .b, c] }
I_{13} = $\sigma(I_5$, b) = { [A → b., c] }
I_{14} = $\sigma(I_8$, A) = { [A → aA., $] }
I_8 = $\sigma(I_8$, a) = { [A → a.A, $] [A → .aA, $] [A → .b, $]}
I_9 = $\sigma(I_8$, b) = { [A → b., $] }
I_{15} = $\sigma(I_{11}$, c) = { [S → cAc., $] }

I_{16} = σ(I_{12} , A) = { [A → aA., c] }
I_{12} = σ(I_{12} , a) et I_{13} = σ(I_{12} , b)

→ Ici, décalage OU réduction : il n'y a pas de conflit → LR(1).

3. Table d'analyse :

	ACTIONS				SUCCESSEURS	
	a	**b**	**c**	**$**	**S**	**A**
0	D3	D4	D5		1	2
1				D6		
2	D8	D9				7
3	D3	D4				10
4	R4	R4				
5	D12	D13				11
6	**ACCEPTATION**					
7				R1		
8	D8	D9				14
9				R4		
10	R3	R3				
11		D15				
12	D12	D13				16
13			R4			
14				R3		
15				R2		
16			R3			

→ on commence par la partie de droite, en ajoutant simplement les items correspondant pour les transitions avec des variables.
→ on s'occupe ensuite de la partie gauche pour laquelle on ajoute les décalages (de la forme D + item correspondant) issus de transition par terminaux.

→ on s'occupe ensuite des réductions, ou il suffit d'ajouter chaque réduction (de la forme R + règle correspondante). Par exemple, on a la réduction suivante : [9,$] = R4. Ceci est dû aux éléments suivants :

- I_9 = $\sigma(I_2$, b) = { [A → b., $] }
- 4. A → b

4. Types de conflits :

- Conflit décaler / réduire
  → un décalage et un réduction en même temps...
- Conflit réduire / réduire
  → deux réductions en même temps...

5. Grammaire SLR(1)
   a. Exercice

0. Analyse du problème :

Considérons la grammaire suivante :

E → E + T | T
T → T x F | F
F → a | b

1. Réécriture de la grammaire :

0. E' → E$
1. E → E + T
2. E → T
3. T → T x F
4. T → F
5. F → a
6. F → b

2. Ensemble Premiers/Suivants :

	PREMIERS	SUIVANTS
E	{a,b}	{$,+}
T	{a,b}	{x,$,+}
F	{a,b}	{x,$,+}
E + T	{a,b}	
T x F	{a,b}	

3. Construction des items :

I_0 = {E' → .E$, E → .E + T, E → .T, T→ .T x F, T → .F, F → .a, F → .b}

I_1 = $\sigma(I_0$, E) = {E' → E.$, E → E. + T}
I_2 = $\sigma(I_0$, F) = {T → F.}
I_3 = $\sigma(I_0$, T) = {E → T., T → T. x F} // Décalage et Réduction → non LR(0)
I_4 = $\sigma(I_0$, a) = {F → a.}
I_5 = $\sigma(I_0$, b) = {F → b.}
I_6 = $\sigma(I_1$, $) = {E' → E$.}
I_7 = $\sigma(I_1$, +) = {E → E + .T, T →.T x F, T →.F, F → .a, F → .b}
I_8 = $\sigma(I_3$, x) = {T →T x. F, F → .a, F → .b}
I_9 = $\sigma(I_7$, T) = {E → E + T., T → T. x F}
I_2 = $\sigma(I_7$, F) = {T → F.}
I_4 = $\sigma(I_7$, a)
I_5 = $\sigma(I_7$, b)
I_{10} = $\sigma(I_8$, F) = {T → T x F.}
I_4 = $\sigma(I_8$, a)
I_5 = $\sigma(I_8$, b)
I_8 = $\sigma(I_9$, x) = {T → T x .F, F → .a, F -> .b}

4. Table d'analyse :

	ACTION					SUCCESSEURS		
	a	**b**	**+**	**x**	**$**	**E**	**T**	**F**
0	D4	D5				1	2	3
1			D7		D6			
2			R4	R4	R4			
3			R2	D8	R2			
4			R5	R5	R5			
5			R6	R6	R6			
6	**ACCEPTATION**							
7	D4	D5					9	2
8	D4	D5						10
9			R1	D8	R1			
10			R3	R3	R3			

→ on commence par la partie de droite, en ajoutant simplement les items correspondant pour les transitions avec des variables.
→ on s'occupe ensuite de la partie gauche pour laquelle on ajoute les décalages (de la forme D + item correspondant) issus de transition par terminaux.

→ enfin, en cas de réduction, il suffit d'ajouter aux suivants de la variable en question, la règle de réduction qui s'applique. Par exemple, I_9 = {E → E + T., T → T. x F} possède une réduction de E vers E + T.. On peut remarquer que cette réduction est issue de la règle R1. Les suivants de E sont {$,+} alors on ajoute en [9,$] et [9,+] la règle R1.

→ ***il y a eu plus une action par cellule, la grammaire est bien SLR(1).***

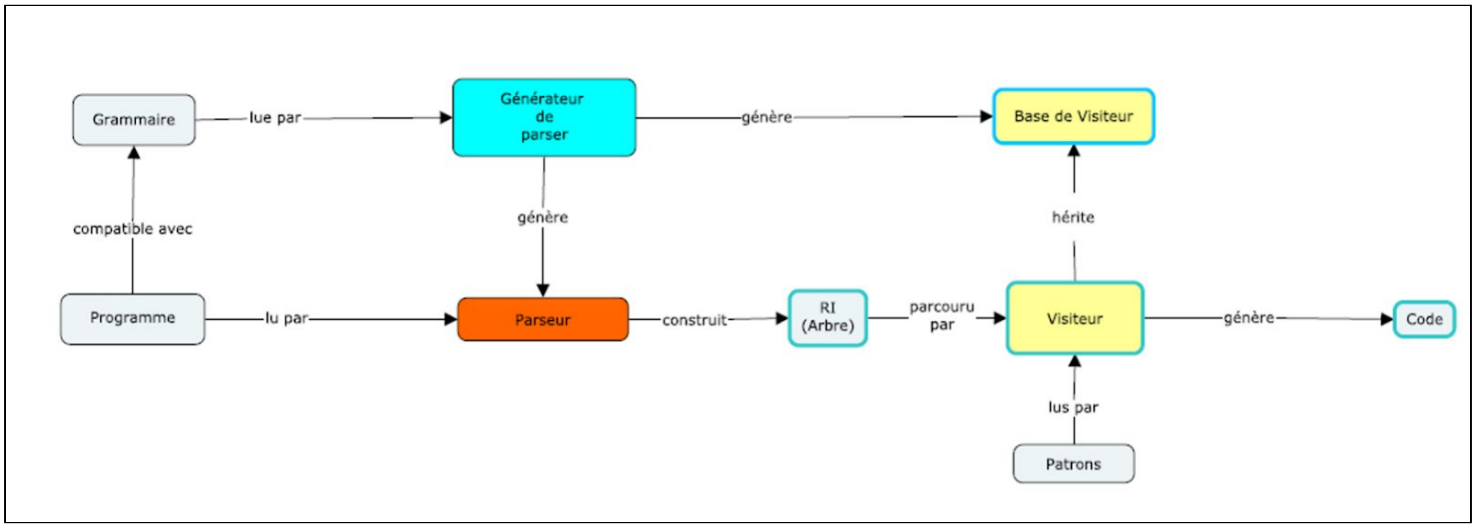
6. Grammaire LALR(0)

→ *On obtient une grammaire LALR(0) en fusionnant les items de LR(0), sans conflit.*
→ *Si une grammaire est LALR(1) alors elle est nécessairement LR(1).*

7. Grammaire AntLR

Éléments nécessaires pour générer un fichier résultat :

- une grammaire
- un générateur (qui produit le code pour lexeur et début visiteur)
- fichier template
- moteur générateur de code



→ imprimer TD13 et le joindre à la suite / comparaison exercice écrit

COURS

(Attribut synthétisé)

Un attribut synthétisé est attaché au non terminal en partie gauche et se calcule en fonction des attributs des symboles de la partie droite. Dans l'arbre décoré, un attribut synthétisé est attaché à un nœud et se calcule en fonction des attributs de ses fils.

(Attribut hérité)

Un attribut est hérité lorsqu'il est calculé à partir des attributs du non terminal de la partie gauche et des attributs des autres symboles de la partie droite. Dans l'arbre décoré, un attribut hérité dépend des attributs du nœud père et des attributs des nœuds frères.

(Définition S-attribuée)
Une définition dirigée par la syntaxe n'ayant que des attributs synthétisés est appelée définition S-attribuée.

(Définition L-attribuée)
Une définition dirigée par la syntaxe est L-attribuée si tout attribut hérité d'un symbole de la partie droite d'une production ne dépend que :

- des attributs hérités du symbole en partie gauche et
- des attributs des symboles le précédant dans la production.

(Définition formelle)
Dans une définition dirigée par la syntaxe, chaque production A → α de la grammaire possède un ensemble de règles sémantiques de la forme : b := f(c1,c2,...ck), où :

- f est une fonction,
- b est soit un attribut synthétisé de A, soit un attribut hérité d'un des symboles figurant en partie droite de la production
- les ci,1≤ i ≤ k sont des attributs quelconques des symboles.

On considère qu'un terminal ne peut avoir que des attributs synthétisés définis à partir des caractères qui composent son lexème. Parfois, il est nécessaire que certaines règles sémantiques produisent des effets de bord. On les considère alors comme des fonctions associées à des attributs factices.

13.2.1 Exemple

Soit la grammaire G définie par les règles suivantes :
$S \longrightarrow aSb/aS/\epsilon$
Il s'agit de déterminer le nombre de a.

Production	Règle sémantique
$S \longrightarrow aSb$	S.nba := S_1.nba + 1
$S \longrightarrow aS$	S.nba := S_1.nba + 1
$S \longrightarrow \epsilon$	S.nba := 0
$S' \longrightarrow S\$$	// S.nba contient la valeur

Dans cet exemple, on s'aperçoit que l'attribut en partie gauche est calculé en fonction de l'attribut en partie droite.

ELEMENTS DE REPONSE

*** **Pas de conflit premier-premier****
(pour chaque non-terminal, les premiers des choix sont tous distincts).

*** **Pas de conflit premier-suivant****
(pour chaque non terminal ayant un choix nullifiable, les premiers et suivants doivent être distincts).

* **Pas de choix nullifiables multiples**.
Il n'y a pas d'éléments (ambigüité, non-factorisation à gauche ou récursivité à gauche) qui signalent directement une impossibilité pour cette grammaire d'être _LL(1)._ On est donc obligé de faire le tableau des premiers et suivants :