Analyse Ascendante

- Précédemment, on construisait l'arbre de la racine vers les feuilles, ici on fait le contraire
- Pour cela, on utilise le modèle Shift-Reduce dans lequel on s'autorise 2 opérations Décalage et Réduction
- Décalage (à droite) du pointeur sur les lettres du mot
- Réduction par remplacement de la partie droite (finissant sur le pointeur) d'une production par sa partie gauche
- Exemple : S → aSbS | c et m= aaacbaacbcbcbcbacbc

Analyse Ascendante : exemple

aaacbaacbcbcbcbacbc → S → aSbS | c ⇒ D.

aaacbaacbcbcbcbacbc ⇒ Décalage

aaacbaacbcbcbcbacbc ⇒ Décalage

aaacbaacbcbcbcbacbc → S → c ⇒ Rédu.

aaaSbaacbcbcbcbacbc ⇒ Décalage

aaaSbaacbcbcbcbacbc ⇒ Décalage

aaaSbaacbcbcbcbacbc ⇒ Décalage

aaaSbaacbcbcbcbacbc → S → c ⇒ Rédu.

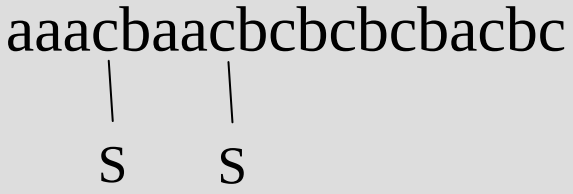
aaaSbaaSbcbcbcbacbc ⇒ Décalage

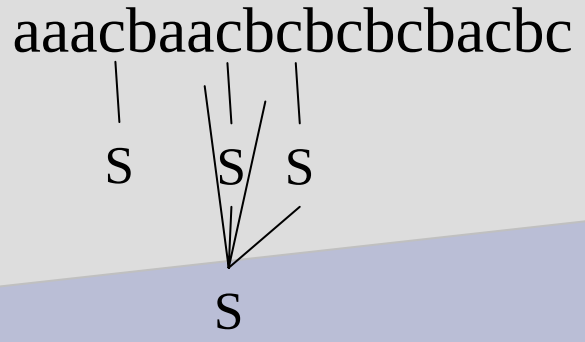
aaaSbaaSbcbcbcbacbc → S → c ⇒ Réd.

aaaSbaaSbSbcbcbacbc → S → aSbS ⇒ R.

aaaSbaSbcbcbacbc ⇒ Décalage

aaaSbaSbcbcbacbc → S → c ⇒ Rédu.





Analyse Ascendante : exemple

aaaSbaSbSbcbacbc → S → aSbS ⇒ réd.

aaaSbSbcbacbc → S → aSbS ⇒ réd.

aaSbcbacbc ⇒ décalage

aaSbcbacbc → S → c ⇒ réduc.

aaSbSbacbc → S → aSbS ⇒ réd.

aSbacbc ⇒ décalage

aSbacbc ⇒ décalage

aSbacbc → S → c ⇒ rédu.

aSbaSbc ⇒ décalage

aSbaSbc ⇒ décalage

aSbaSbc → S → c ⇒ rédu.

aSbaSbS → S → aSbS ⇒ réd.

aSbS → S → aSbS ⇒ réd.

S ⇒ Accepté

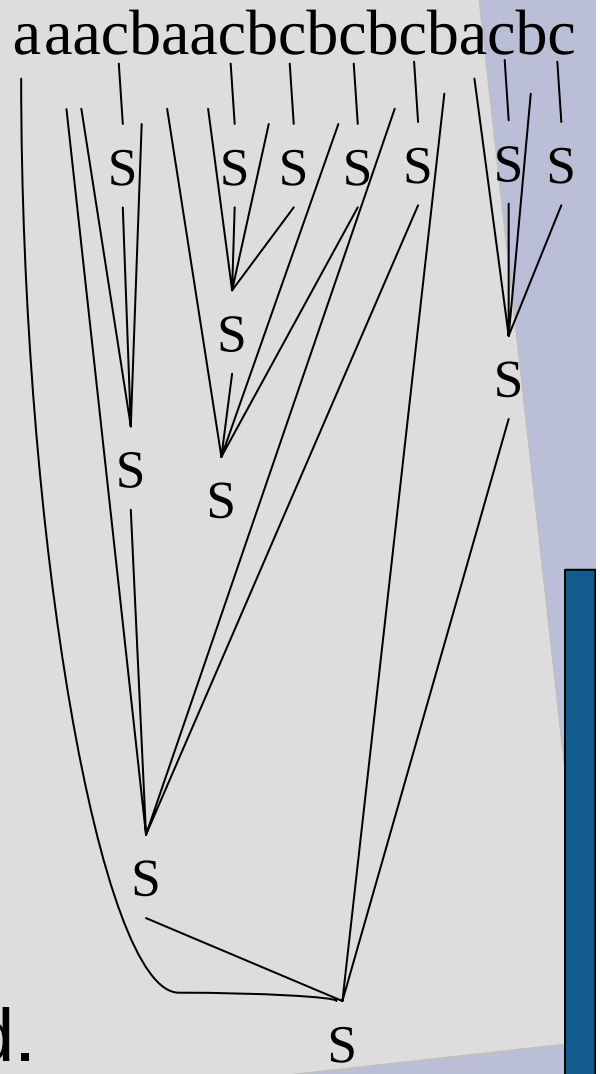


Table d'analyse SLR

- Dans l'exemple précédent, il y avait peu de production. Il était donc facile de trouver la production à appliquer (réduction). Dans le cas général, on sera amené à construire une table comme on l'a fait pour l'analyse descendante.
- Ces analyseurs syntaxiques qui lisent l'entrée de gauche à droite et produisent une dérivation droite, sont appelés analyseurs syntaxiques LR simples (SLR pour "Simple LR parser"), analyseurs syntaxiques LR avec anticipation (LALR pour "Look-Ahead LR parser"), et analyseurs syntaxiques canoniques. Ces types d'analyseurs syntaxiques peuvent traiter des ensembles de grammaires de plus en plus grands ; les analyseurs syntaxiques LALR peuvent traiter plus de grammaires que les SLR. Les analyseurs syntaxiques canoniques fonctionnent sur davantage de grammaires que les analyseurs syntaxiques LALR.

Construction de table d'analyse syntaxique SLR

- **Items :** La construction de ces tables d'analyse syntaxique est basée sur la notion d'*items LR(0)* ou 0-items (simplement appelés *items* ici) qui sont des règles de grammaire avec un point spécial ajouté quelque part dans la partie droite. Par exemple, la règle E → E + B a quatre items correspondants :
  - E → • E + B
  - E → E • + B
  - E → E + • B
  - E → E + B •

Exemple G:
E → E + B | E * B | B
B → 0 | 1 (5 règles)

Construction de table d'analyse syntaxique SLR

- Les règles de la forme $A \rightarrow \varepsilon$ ont un seul item $A \rightarrow \bullet$. Ces règles seront utilisées pour indiquer l'état de l'analyseur syntaxique. L'item $E \rightarrow E \bullet + B$, par exemple, indique que l'analyseur syntaxique a reconnu une chaîne correspondant à E sur le flot d'entrée, et qu'il attend un '+' suivi d'une autre chaîne correspondant à B.
- **Ensemble d'items :** Il n'est pas toujours possible de caractériser l'état de l'analyseur syntaxique avec un seul item. En effet, il peut arriver qu'on ne sache pas d'avance quelle règle va être utilisée pour la réduction. Dans notre exemple, après la lecture d'une chaîne correspondant à un E, il y a deux items candidats à la réduction : $E \rightarrow E \bullet * B$ et $E \rightarrow E \bullet + B$. Par conséquent, on caractérisera l'état par un *ensemble* d'items, c'est-à-dire $\{ E \rightarrow E \bullet + B, E \rightarrow E \bullet * B \}$ dans ce cas.

Construction de table d'analyse syntaxique SLR

- **Fermeture des ensembles d'items** Un item avec un point devant un non-terminal, tel que $E \rightarrow E + \bullet B$, indique que l'analyseur syntaxique s'attend à trouver ce non-terminal (en l'occurrence B, dans cet exemple). Pour s'assurer que l'ensemble d'items contient toutes les règles possibles, l'analyseur syntaxique doit inclure tous les items décrivant comment ce non-terminal B sera analysé. Cela signifie que s'il y a des règles telles que $B \rightarrow 0$ et $B \rightarrow 1$, l'ensemble d'items doit aussi inclure $B \rightarrow \bullet 0$ et $B \rightarrow \bullet 1$.

Construction de table d'analyse syntaxique SLR

- De manière générale, cela peut être formulé ainsi :
  - S'il y a un item de la forme $A \rightarrow \alpha \bullet B\beta$ dans l'ensemble d'items, et s'il y a une règle de la forme $B \rightarrow \gamma$ alors l'item $B \rightarrow \bullet \gamma$ doit également se trouver dans l'ensemble d'items.
- Tous les ensembles d'items peuvent être étendus jusqu'à ce qu'ils satisfassent cette règle : continuer simplement d'ajouter les items appropriés jusqu'à ce que tous les non-terminaux précédés par des points soient pris en compte. L'extension minimale est appelée la *fermeture* d'un ensemble d'items et on l'écrira **ferm**(*I*) où *I* est un ensemble d'items. Ce sont ces ensembles d'items fermés qui seront considérés dans les états de l'analyseur syntaxique, bien que seuls ceux qui sont réellement accessibles à partir de l'état initial seront inclus dans les tables.

Construction de table d'analyse syntaxique SLR

- **La grammaire augmentée** Avant de commencer à déterminer les transitions entre les différents états, la grammaire est augmentée de la règle suivante : (0) S → E
- où S est le nouveau symbole de départ et E l'ancien. L'analyseur syntaxique utilisera cette règle pour la réduction exactement quand il aura accepté la chaîne d'entrée.
- Pour notre exemple, nous prendrons la même grammaire avec cette augmentation. Soit :
  - (0) S → E
  - (1) E → E * B
  - (2) E → E + B
  - (3) E → B
  - (4) B → 0
  - (5) B → 1

Exemple G :

(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Construction de table d'analyse syntaxique SLR

- La première étape de la construction des tables consiste à déterminer la transition entre les ensembles d'items fermés. Ces transitions seront déterminées comme si on considérait un automate fini qui pourrait lire des terminaux et des non-terminaux. L'état initial de cet automate est la fermeture du premier item de la règle ajoutée : $S \rightarrow \bullet E$:
  - **Ensemble d'items 0**
  - $S \rightarrow \bullet E$
  - $+ E \rightarrow \bullet E * B$
  - $+ E \rightarrow \bullet E + B$
  - $+ E \rightarrow \bullet B$
  - $+ B \rightarrow \bullet 0$
  - $+ B \rightarrow \bullet 1$

$$I_0 = \{S \rightarrow \bullet E,\ E \rightarrow \bullet E * B,\ E \rightarrow \bullet E + B,\ E \rightarrow \bullet B,\ B \rightarrow \bullet 0,\ B \rightarrow \bullet 1\}$$

- Le '+' devant les items indique ceux qui ont été ajoutés pour la fermeture. Les items originaux sans '+' sont appelés le *noyau* de l'ensemble d'items.

Construction de table d'analyse syntaxique SLR

- En partant de l'état initial (S0), on déterminera tous les états qui peuvent être atteints en une étape. Les transitions possibles pour un ensemble d'items peuvent être trouvés en regardant les symboles (terminaux et non-terminaux) qui se trouvent juste après le point. Dans le cas de l'ensemble d'items 0, ce sont les terminaux '0' et '1' et les non-terminaux E et B. Pour trouver l'ensemble d'items auxquels conduit un symbole *x*, on suit la procédure suivante :
- Prendre l'ensemble *S* de tous les items où il y a un point en face de ce symbole *x*.
- Pour chaque item dans *S*, déplacer le point à la droite de *x*.
- Compléter par fermeture l'ensemble des items.

Construction de table d'analyse syntaxique SLR

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

$I_0 = \{S \rightarrow \bullet E, E \rightarrow \bullet E * B, E \rightarrow \bullet E + B, E \rightarrow \bullet B, B \rightarrow \bullet 0, B \rightarrow \bullet 1\}$

- Pour le terminal '0', le résultat est :
  - **Ensemble d'items 1 :** $I_1 = \Delta(I_0,0)$
  - $B \rightarrow 0 \bullet$
- et pour le terminal '1' :
  - **Ensemble d'items 2 :** $I_2 = \Delta(I_0,1)$
  - $B \rightarrow 1 \bullet$
- Pour le non-terminal E :
  - **Ensemble d'items 3 :** $I_3 = \Delta(I_0,E)$
  - $S \rightarrow E \bullet$
  - $E \rightarrow E \bullet * B$
  - $E \rightarrow E \bullet + B$
- et pour le non-terminal B :
  - **Ensemble d'items 4:** $I_4 = \Delta(I_0,B)$
  - $E \rightarrow B \bullet$

- La fermeture n'ajoute pas de nouveaux items dans tous les cas. On continue la procédure jusqu'à ce qu'aucun autre ensemble d'items puisse être trouvé. Pour les ensembles d'items 1, 2 et 4, il n'y aura pas de transition puis qu'aucun point ne se trouve devant un symbole.

Construction de table d'analyse syntaxique SLR

- Pour l'ensemble d'items 3, on voit que le point se trouve devant les terminaux '*' et '+'.
  Pour '*', la transition donne :
  - **Ensemble d'items 5 :** $I_5 = \Delta(I_3,*)$
  - $E \rightarrow E * \bullet B$
  - $+ B \rightarrow \bullet 0$
  - $+ B \rightarrow \bullet 1$
- et pour '+' elle donne :
  - **Ensemble d'items 6 :** $I_6 = \Delta(I_3,+)$
  - $E \rightarrow E + \bullet B$
  - $+ B \rightarrow \bullet 0$
  - $+ B \rightarrow \bullet 1$

$I_3 = \{S \rightarrow E \bullet, E \rightarrow E \bullet * B, E \rightarrow E \bullet + B\}$

•Pour l'ensemble d'items 5, on doit considérer les terminaux '0' et '1' et le non-terminal B. Pour les terminaux, on voit que les ensembles d'items fermés sont égaux respectivement aux ensembles d'items 1 et 2 déjà construits.

Construction de table d'analyse syntaxique SLR

- Pour le non-terminal B, la transition donne :
  - **Ensemble d'items 7 :** $I_7 = \Delta(I_5,B)$
  - $E \rightarrow E * B \bullet$

$I_5 = \{E \rightarrow E * \bullet B, B \rightarrow \bullet 0, B \rightarrow \bullet 1\}$
$I_6 = \{E \rightarrow E + \bullet B, B \rightarrow \bullet 0, B \rightarrow \bullet 1\}$

- Pour l'ensemble d'items 6, il faut également considérer les terminaux '0' et '1' et le non-terminal B. Comme précédemment, les ensembles d'items résultats pour les terminaux sont égaux aux ensembles d'items 1 et 2 déjà construits : $\Delta(I_6,0)=I_2$. Pour le non-terminal B, la transition conduit à :
  - **Ensemble d'items 8 :** $I_8 = \Delta(I_6,B)$
  - $E \rightarrow E + B \bullet$
- Ces ensembles d'items 7 et 8 n'ont pas de symboles derrière leurs points et, donc, c'est fini.

Construction de table d'analyse syntaxique SLR

- La table de transitions $I_j = \Delta(I_i,\alpha)$ pour l'automate ressemble à : ⟹

I_0) $S \rightarrow \bullet E$, $E \rightarrow \bullet E * B$, $E \rightarrow \bullet E + B$, $E \rightarrow \bullet B$, $B \rightarrow \bullet 0$, $B \rightarrow \bullet 1$

I_1) $B \rightarrow 0 \bullet$

I_2) $B \rightarrow 1 \bullet$

I_3) $S \rightarrow E \bullet$, $E \rightarrow E \bullet * B$, $E \rightarrow E \bullet + B$

I_4) $E \rightarrow B \bullet$

I_5) $E \rightarrow E * \bullet B$, $B \rightarrow \bullet 0$, $B \rightarrow \bullet 1$

I_6) $E \rightarrow E + \bullet B$, $B \rightarrow \bullet 0$, $B \rightarrow \bullet 1$

I_7) $E \rightarrow E * B \bullet$

I_8) $E \rightarrow E + B \bullet$

(0) $S \rightarrow E$
(1) $E \rightarrow E * B$
(2) $E \rightarrow E + B$
(3) $E \rightarrow B$
(4) $B \rightarrow 0$
(5) $B \rightarrow 1$

Ensemble d'items	*	+	0	1	E	B
0			1	2	3	4
1						
2						
3	5	6				
4						
5			1	2		7
6			1	2		8
7						
8						

Construction de table d'analyse syntaxique SLR

(Soit S l'axiome initial, la grammaire est augmentée de S' →S)

- pour chaque A → α• sauf S' → α• contenu dans :
  pour chaque a de SUIVANT(A) faire
  mettre réduction par numéro (de la règle) dans la case T[i,a]
- Si S' → S•∈I_i mettre ACCepter dans la case T[i,$]

	Prem	Suiv
S	0,1	$
E	0,1	$,+,*
B	0,1	$,+,*

Construction de table d'analyse syntaxique SLR

I_0) $S \rightarrow \bullet E$, $E \rightarrow \bullet E * B$,
$E \rightarrow \bullet E + B$, $E \rightarrow \bullet B$,
$B \rightarrow \bullet 0$, $B \rightarrow \bullet 1$

I_1) $B \rightarrow 0 \bullet$

I_2) $B \rightarrow 1 \bullet$

I_3) $S \rightarrow E \bullet$, $E \rightarrow E \bullet * B$,
$E \rightarrow E \bullet + B$

I_4) $E \rightarrow B \bullet$

I_5) $E \rightarrow E * \bullet B$, $B \rightarrow \bullet 0$,
$B \rightarrow \bullet 1$

I_6) $E \rightarrow E + \bullet B$, $B \rightarrow \bullet 0$,
$B \rightarrow \bullet 1$

I_7) $E \rightarrow E * B \bullet$

I_8) $E \rightarrow E + B \bullet$

« sauf S' → α• …»
« Si S' → S•∈Ii…»

	Prem	Suiv
S	0,1	$
E	0,1	$,+,*
B	0,1	$,+,*

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

	*	+	0	1	$
0					
1	4	4			4
2	5	5			5
3					Acc
4	3	3			3
5					
6					
7	1	1			1
8	2	2			2

Construction de table d'analyse syntaxique SLR

- En regroupant le premier tableau dans lequel les états terminaux nous donnait les décalages (shift) et les non-terminaux nous donnait les branchements avec le second tableau qui nous donne, pour les terminaux et le $, les réductions et acceptations

Ensemble d'items	*	+	0	1	E	B
0			s1	s2	3	4
1						
2						
3	s5	s6				
4						
5			s1	s2		7
6			s1	s2		8
7						
8						

	*	+	0	1	$
0					
1	r4	r4			r4
2	r5	r5			r5
3					Acc
4	r3	r3			r3
5					
6					
7	r1	r1			r1
8	r2	r2			r2

Construction de table d'analyse syntaxique SLR

- En regroupant le premier tableau dans lequel les états terminaux nous donnait les décalages (shift) et les non-terminaux nous donnait les branchements avec le second tableau qui nous donne, pour les terminaux et le $, les réductions et acceptations

état	*action*						*branchement*	
	*	+	0	1	$		E	B
0			s1	s2			3	4
1	r4	r4			r4			
2	r5	r5			r5			
3	s5	s6			acc			
4	r3	r3			r3			
5			s1	s2				7
6			s1	s2				8
7	r1	r1			r1			
8	r2	r2			r2			

Utilisation de table d'analyse syntaxique SLR

- Quand l'analyseur syntaxique démarre, il part toujours avec l'état 0 dans la pile :[**0**]
- Soit à analyser 1+1$
- Le premier terminal que l'analyseur syntaxique voit est le '1'. D'après la table des actions, il doit aller à l'état 2. Ce qui donne pour la pile :
  - [**0** '1' **2**]

état	*action*						*branchement*	
	*	+	0	1	$		E	B
0			s1	s2			3	4
1	r4	r4			r4			
2	r5	r5			r5			
3	s5	s6			acc			
4	r3	r3			r3			
5			s1	s2				7
6			s1	s2				8
7	r1	r1			r1			
8	r2	r2			r2			

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	

état	action					branchement	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

▼

1 + 1

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

1 ▼ + 1

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

1 + 1
|
B

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

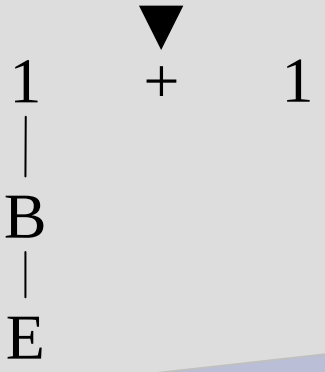
1 + 1
|
B

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

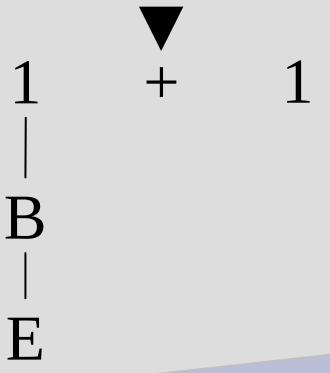


(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		



(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3
$0E3	+ 1 $	s6

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

1 | B | E

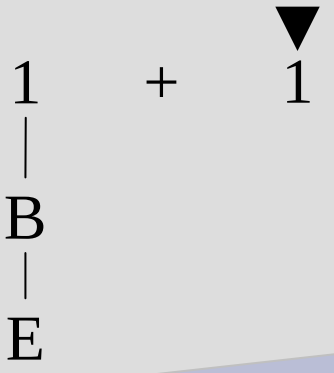
▼ + 1

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3
$0E3	+ 1 $	s6
$0E3+6	1 $	s2

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		



(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3
$0E3	+ 1 $	s6
$0E3+6	1 $	s2
$0E3+612	$	

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

1 + 1$

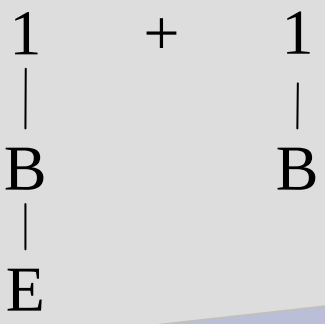
1
|
B
|
E

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3
$0E3	+ 1 $	s6
$0E3+6	1 $	s2
$0E3+612	$	r5 : B →1

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

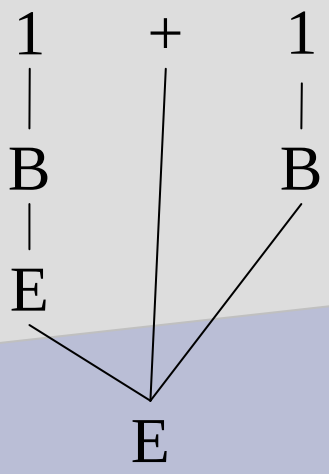


(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3
$0E3	+ 1 $	s6
$0E3+6	1 $	s2
$0E3+612	$	r5 : B →1
$0E3+6B	$	État 6 avec B : aller en 8
$0E3+6B8	$	r2 : E →E + B

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		

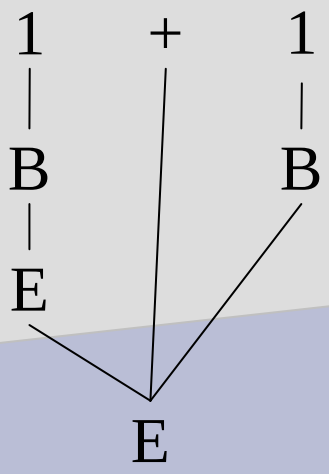


(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Pile : En rouge, les États ; En noir, les T et NT

pile	entrée	Action
$0	1 + 1 $	s2
$012	+ 1 $	r5 : B →1
$0B	+ 1 $	État 0 avec B : aller en 4
$0B4	+ 1 $	r3 : E →B
$0E	+ 1 $	État 0 avec E : aller en 3
$0E3	+ 1 $	s6
$0E3+6	1 $	s2
$0E3+612	$	r5 : B →1
$0E3+6B	$	État 6 avec B : aller en 8
$0E3+6B8	$	r2 : E →E + B
$0E	$	État 0 avec E : aller en 3
$0E3	$	ACCepté

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4			r4		
2	r5	r5			r5		
3	s5	s6			acc		
4	r3	r3			r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1			r1		
8	r2	r2			r2		



(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Remarques sur l'analyse syntaxique SLR

- Cette méthode permet d'analyser plus de grammaires que la méthode descendante (car il y a plus de grammaires SLR que LL)
- Contrairement à l'analyse descendante, il est préférable que la grammaire soit récursive à gauche.
- Les grammaires ambiguës provoquent des **conflits**
  - conflit décalage/réduction : on ne peut pas décider à la lecture du terminal *a* s'il faut réduire une production $S \rightarrow \alpha$ ou décaler le terminal
  - conflit réduction/réduction : on ne peut pas décider à la lecture du terminal *a* s'il faut réduire une production $S \rightarrow \alpha$ ou une production $T \rightarrow \beta$

Conflits Shift / Reduce

On doit alors résoudre les conflits en donnant des **priorités** aux actions (décaler ou réduire) et aux productions.

Par exemple, soit la grammaire E → E + E | E x E | (E) | nb

Soit à analyser 3+4+5. A la lecture du 2ème +, on a le choix entre

- réduire ce qu'on a déjà lu par E → E + E . Ce qui nous donnera finalement le calcul (3+4)+5
- décaler ce +, ce qui nous donnera finalement le calcul 3+(4+5).

+ est associatif, cela n'a aucune importance, donc on préfèrera réduire.

Soit à analyser 3+4*5. Lorsqu'on lit le * on a encore un choix shift/reduce. Si l'on réduit on calcule (3+4)*5, si on décale on calcule 3+(4*5) ! Ici ce n'est plus pareil ! Il **faut** décaler.
Soit à analyser 3*4+5. là encore le choix est important, il faut réduire !
Bref, il faut mettre quelque part dans l'analyseur le fait que * est prioritaire sur +.

Erreurs syntaxiques

- Beaucoup d'erreurs sont par nature syntaxiques (ou révélées lorsque les unités lexicales provenant de l'analyseur lexical contredisent les règles grammaticales).
  Le gestionnaire d'erreur doit
  - indiquer la présence de l'erreur de façon claire et précise
  - traiter l'erreur rapidement pour continuer l'analyse
  - traiter l'erreur le plus efficacement possible de manière à ne pas en créer de nouvelles.
  Heureusement, les erreurs communes (confusion entre deux séparateurs (par exemple entre ; et ,), oubli de ;, ...) sont simples et un mécanisme simple de traitement suffit en général. Cependant, une erreur peut se produire longtemps avant d'être détectée (par exemple l'oubli d'un { ou } dans un programme C). La nature de l'erreur est alors très difficile à déduire. La plupart du temps, le gestionnaire d'erreurs doit **deviner** ce que le programmeur avait en tête.
  Lorsque le nombre d'erreur devient trop important, il est plus raisonnable de stopper

Erreurs syntaxiques

- Il existe plusieurs stratégies de récupération sur erreur : mode panique, au niveau du syntagme, productions d'erreur, correction globale. Une récupération inadéquate peut provoquer une avalanche néfastes d'erreurs *illégitimes*, c'est à dire d'erreurs qui n'ont pas été faites par le programmeur mais sont la conséquence du changement d'état de l'analyseur lors de la récupération sur erreur. Ces erreurs illégitimes peuvent être syntaxiques mais également sémantiques. Par exemple, pour se récupérer d'une erreur, l'analyseur syntaxique peut sauter la déclaration d'une variable. Lors de l'utilisation de cette variable, l'analyseur sémantique indiquera qu'elle n'a pas été déclarée.

Erreurs syntaxiques : Récupération en mode panique

- C'est la méthode la plus simple à implanter. Quand il découvre une erreur, l'analyseur syntaxique élimine les symboles d'entrée les uns après les autres jusqu'à en rencontrer un qui appartienne à un ensemble d'unités lexicales de synchronisation, c'est à dire (par exemple) les délimiteurs (;, end ou }), dont le rôle dans un programme source est clair. Bien que cette méthode saute en général une partie considérable du texte source sans en vérifier la validité, elle a l'avantage de la simplicité et ne peut pas entrer dans une boucle infinie.

Erreurs syntaxiques : Récupération au niveau du syntagme

- Quand une erreur est découverte, l'analyseur syntaxique peut effectuer des corrections locales. Par exemple, remplacer une , par un ;, un wihle par un while, insérer un ; ou une (, ...
- Le choix de la modification à faire n'est pas évident du tout, en général. En outre, il faut faire attention à ne pas faire de modifications qui entraînerait une boucle infinie (par exemple décider d'insérer systématiquement un symbole juste avant le symbole courant).
  L'inconvénient majeure de cette méthode est qu'il est pratiquement impossible de gérer les situations dans lesquelles l'erreur réelle s'est produite bien avant le point de détection.

Erreurs syntaxiques : Récupération au niveau du syntagme

- On implante cette récupération sur erreur en remplissant les cases vides des tables d'analyse par des pointeurs vers des routines d'erreur. Ces routines remplacent, insèrent ou suppriment des symboles d'entrée et émettent les messages appropriés.
- Dans l'exemple de $E \rightarrow E * B \mid E + B \mid B$ et $B \rightarrow 0 \mid 1$

Traitement d'erreurs

G : {E → E * B | E + B | B , B → 0 | 1}

I_0) S → • E, E → • E * B, E → • E + B,
E → • B, B → • 0, B → • 1

I_1) B → 0 •

I_2) B → 1 •

I_3) S → E •, E → E • * B, E → E • + B

I_4) E → B •

I_5) E → E * • B, B → • 0, B → • 1

I_6) E → E + • B, B → • 0, B → • 1

I_7) E → E * B •

I_8) E → E + B •

état	*action*					*branchement*	
	*	+	0	1	$	E	B
0	e1	e1	s1	s2	e1	3	4
1	r4	r4	e4	e4	r4		
2	r5	r5	e4	e4	r5		
3	s5	s6	e2	e2	acc		
4	r3	r3	e4	e4	r3		
5	e3	e3	s1	s2	e1		7
6	e3	e3	s1	s2	e1		8
7	r1	r1	e4	e4	r1		
8	r2	r2	e4	e4	r2		

e1 : Opérande attendue : ajout de 1 (ou 0) et poursuite de l'analyse
e2 : Opérateur attendu : ajout de + (ou *) et poursuite de l'analyse
e3 : 2 opérateurs ne peuvent se suivre : suppression 1 opérateur et poursuite
e4 : 2 opérandes ne peuvent se suivre : suppression 1 opérande et poursuite
On pourrait ainsi remplir toutes les cases vides : pas tjr évident !