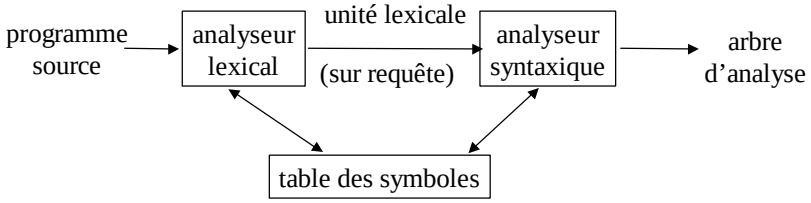
CH.2 L'ANALYSE SYNTAXIQUE

- 2.1 Le rôle de l'analyseur syntaxique
- 2.2 Les grammaires
- 2.3 L'analyse descendante
- 2.4 L'analyse ascendante

2.1 Le rôle de l'analyseur syntaxique



Rôle central de la partie frontale :
- active l'analyseur lexical;
- vérifie la conformité syntaxique;
- construit l'**arbre d'analyse** ;
- prépare ou anticipe la traduction ;
- gère les erreurs communes de syntaxe.

Types d'analyseurs syntaxiques :

- méthodes universelles (Cocke-Younger-Kasami), permettent une analyse de n'importe quelle grammaire algébrique, mais performance en $O(n^3)$;
- méthodes linéaires en $O(n)$, sur certaines grammaires :
  analyse ascendante, la plus intuitive, se prête bien à certains types de traductions ;
  analyse descendante, plus sophistiquée, la plus utilisée par les générateurs automatiques d'analyseurs syntaxiques, car ne nécessite que peu d'adaptations de la grammaire.

Traitement des erreurs :

- diagnostic (messages) ;
- redémarrage :
  mode panique, jusqu'à resynchronisation ;
  correction, difficile si l'erreur est antérieure à sa détection ;
  règles d'erreurs, intégrées à la grammaire.

2.2 Les grammaires

Syntaxe spécifiée par des **règles de grammaire**.

- Symboles terminaux (= unités lexicales) alphabet A ;
- Symboles intermédiaires ou variables (= catégories grammaticales) alphabet X ;
- Règles de grammaire $x \rightarrow w$, où $x \in X$ et où $w \in (A \cup X)^*$
  w est un mot quelconque, même vide.

Exemples :

instr → **si** *expr* **alors** *instr* **sinon** *instr*
phrase → *gsujet gverbe gcomplément*

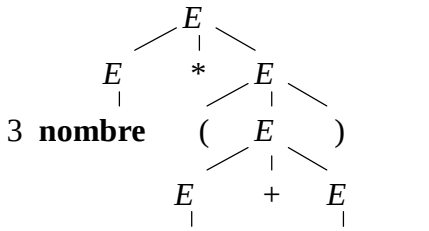
- Axiome (= *programme*)

Langage engendré = mots terminaux dérivant de l'axiome.

Exemple : Expressions arithmétiques

$E \rightarrow$ **nombre**
$E \rightarrow (E)$
$E \rightarrow E + E$
$E \rightarrow E - E$
$E \rightarrow E * E$
$E \rightarrow E / E$

Arbre d'analyse de 3*(6+2)



Les valeurs explicites sont des attributs des unités lexicales.
L'association aux unités lexicales est réalisée par l'analyseur lexical.

Grammaire précédente ambiguë

Grammaire non ambiguë pour expressions arithmétiques suffixes :

$E \rightarrow EE\ +\ |\ EE\ -\ |\ EE\ *\ |\ EE\ /\ |$ **nombre**

Grammaire non ambiguë pour les expressions arithmétiques :

$E \rightarrow E + T\ |\ E - T\ |\ T$
$T \rightarrow T * F\ |\ T / F\ |\ F$
$F \rightarrow (E)\ |$ **nombre**

Choix : associativité à gauche
priorité de * et / sur + et –

Analyse syntaxique :
Étant donné un mot terminal, déterminer s'il est ou non engendré par la grammaire ; si oui, en donner un arbre d'analyse.

- Méthode universelle : essayer toutes les dérivations à partir de l'axiome, jusqu'à trouver le mot. Des règles de longueur (après modifications) permettent d'éliminer les impasses.

- Méthode descendante (descente récursive) : une procédure par variable, les terminaux servant à choisir la dérivation (prévision) et à la validation.

- Méthode ascendante : on lit le mot (décalage) jusqu'à identifier des dérivation, qu'on réduit et empile (réduction).

2.3 L'analyse descendante

Une procédure par terme.
Problème : récursivité directe à gauche ;
nécessité d'une transformation pour l'éliminer.

Grammaire de départ

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid$ **nombre**

Grammaire modifiée

$A \rightarrow E\$$ **(production augmentée)**
$E \rightarrow TG$
$G \rightarrow +TG \mid e$
$T \rightarrow FU$
$U \rightarrow *FU \mid e$
$F \rightarrow (E) \mid$ **nombre**

Procédures correspondantes : deux procédures supposées écrites :
- prevision retourne l'unité lexicale suivante sans l'enlever ;
- correspond(x) lit l'unité lexicale suivante, l'enlève et signale une erreur si elle ne vaut pas x. Ici plus, nombre, etc. sont les codes d'unités lexicales renvoyés par l'analyseur lexical.

```
procedure expression ;
  begin  writeln('E->TG') ;
    terme ; encoreterme
  end ;

procedure encoreterme ;
  begin
    if prevision = plus then
      begin writeln('G->+TG') ;
        correspond(plus) ;
        terme ; encoreterme
      end
    else  writeln('G->epsilon')
  end ;
```

```
procedure expression ;
  begin writeln('E->TG') ;
    terme ;
    while prevision = plus do
      begin
        writeln('G->+TG') ;
        correspond(plus) ;
        terme
      end ;
    writeln('G->epsilon')
  end ;
```

En éliminant la récursivité terminale et en regroupant

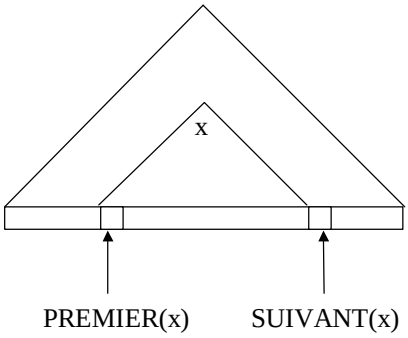

```
procedure terme ;
  begin  writeln('T->FU');
    facteur ;
    while prevision = mult do
      begin  writeln('U->*FU') ;
        correspond(mult) ;
        facteur
      end ;
    writeln('U->epsilon')
  end ;
```

```
procedure facteur ;
  begin
    if prevision = ouvrante then
      begin writeln('F->(E)') ;
        correspond(ouvrante) ;
        expression ;
        correspond(fermante)
      end
    else if prevision = nombre then
      begin writeln('F->nombre');
        correspond(nombre)
      end
    else writeln('erreur syntaxique')
  end ;
```

```
procedure analyse ;
  begin writeln('A->E$') ;
    expression ; correspond(dollar) ; writeln('analyse reussie')
  end ;
```

L'échec de l'analyse est aussi pris en charge par correspond(x).

Fonctions PREMIER et SUIVANT



PREMIER(x) = ensemble des terminaux pouvant apparaître au début d'une dérivation de x.

SUIVANT(x) = ensemble des terminaux pouvant suivre x dans une dérivation

Calcul de PREMIER :

On construit un graphe entre tous les symboles grammaticaux.
Flèche de x vers y ssi $x \rightarrow a\ yb$, et a est annulable.
Ici, a ne peut contenir que des variables, b est quelconque.
PREMIER(x) = { a terminal | il existe un chemin de x à a } ;
si x est annulable, il faut y ajouter e.

Exemple de la grammaire précédente : Annulables : G et U

Graphe : $A \rightarrow E \rightarrow T \rightarrow F \rightarrow$ ($G \rightarrow$ + $U \rightarrow$ *

nombre

PREMIER(A) = PREMIER(E) = PREMIER(T) = PREMIER(F) = { (, **nombre** }
PREMIER(G) = {+, e } PREMIER(U) = { *, e }

Calcul de SUIVANT :

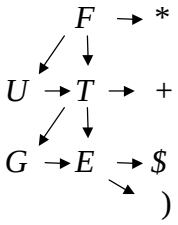
On construit un graphe entre tous les symboles grammaticaux.
Flèche de x vers y ssi ou bien

- il existe $z \rightarrow a\ xb$, y est terminal et $y \in$ PREMIER(b) ;
- il existe $y \rightarrow a\ xb$ et b est annulable.

Ici, a et b sont quelconques, même vides, et $y \neq e$.
SUIVANT(x) = { a terminal | il existe un chemin de x à a }.

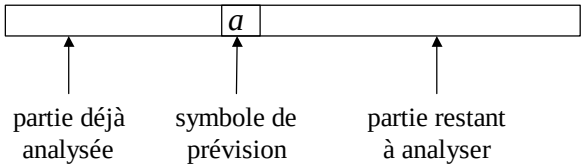
Exemple de la grammaire précédente :

Graphe :



SUIVANT n'a pas de sens pour l'axiome A de la production augmentée.
SUIVANT(G) = SUIVANT(E) = { $,) }
SUIVANT(U) = SUIVANT(T) = { +, $,) }
SUIVANT(F) = { *, +, $,) }

PREMIER et SUIVANT permettent de caractériser les grammaires analysables de façon descendantes, avec un seul symbole de prévision a :



Grammaires *LL*(1).

Si a non annulable, règle $x \rightarrow a$ appliquée lorsque $a \in$ PREMIER(a) ;
Si a annulable, règle $x \rightarrow a$ appliquée lorsque $a \in$ SUIVANT(x).

La grammaire est ***LL*(1)** lorsqu'à chaque étape, une seule règle satisfait aux critères précédents.

Table d'analyse *LL*(1) :

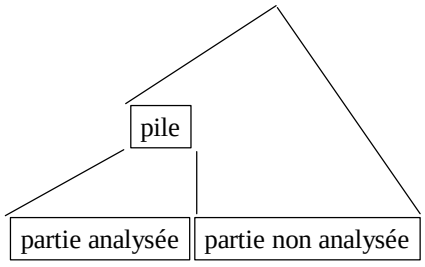
	+	*	(	)	**n**	*$*
E			1		1	
G	2			3		3
T			4		4	
U	6	5		6		6
F			7		8	

0 $A \rightarrow E\$$
1 $E \rightarrow TG$
2 $G \rightarrow +TG$
3 $G \rightarrow e$
4 $T \rightarrow FU$
5 $U \rightarrow *FU$
6 $U \rightarrow e$
7 $F \rightarrow (E)$
8 $F \rightarrow$ **nombre**

Les case vides de la table correspondent à des erreurs syntaxiques.
La table commande les branchements dans les procédures de l'analyse par descente récursive.
On peut l'utiliser comme donnée d'un programme universel d'analyse descendante non récursive.

2.4 L'analyse ascendante

On utilise une pile, et une table de transition entre états.
(Automate déterministe à une pile)
En fonction du symbole de prévision,
- on empile un état en lisant un caractère de l'entrée (décalage) ;
- on dépile autant de symboles que la longueur de la règle qu'on a reconnue, et on empile un nouvel état (réduction).



Exemple : expressions arithmétiques

0 $A \to E\$$
1 $E \to E + T$
2 $E \to T$
3 $T \to T * F$
4 $T \to F$
5 $F \to (E)$
6 $F \to \mathbf{n}$

PREMIER :

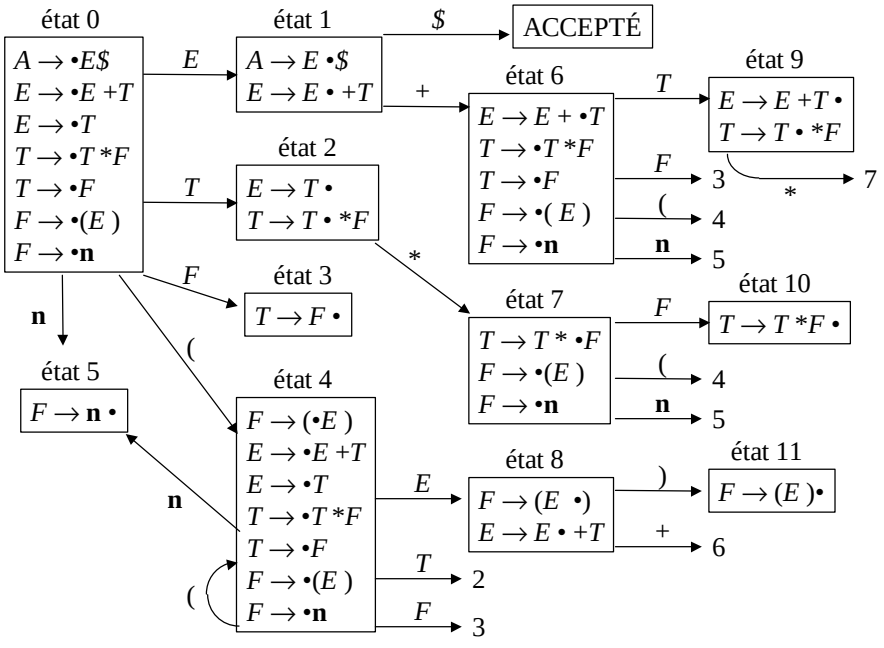
$A \to E \to T \to F \to ($
$\mathbf{n}$

SUIVANT :

$F \to T \to E \to \$$
$+$
$*$ $)$

Calcul des états : ensembles de règles marquées (items) ;

Si un état contient $x \to u \bullet yv$, il contient aussi tous les $y \to \bullet w$ (clôture) ;
Si un état contient $x \to u \bullet yv$, on a une transition par y sur un état contenant $x \to uy \bullet v$.



Fonctionnement de l'analyseur :

Au début, la pile contient l'état 0.
On lit la lettre courante du mot à analyser ; avec l'état du sommet de pile, elle spécifie l'état à empiler **(décalage)**.
Lorsqu'un état contient un item du type $x \to u \bullet$, on peut enlever de la pile les $|u|$ états au sommet, et empiler l'état spécifié par le nouveau sommet et x **(réduction)**.

Parfois des conflits se produisent : décalage–réduction ou réduction–réduction ; cas général, $x \to u \bullet$, $y \to v \bullet$, $z \to r \bullet st$, où s est est terminale. Le conflit peut être tranché lorsque SUIVANT(x), SUIVANT(y) et s sont disjoints, en examinant le caractère de pré-vision.

Une telle grammaire est dite ***SLR(1)***.

La suite des réductions constitue une dérivation du mot à analyser, à partir de la fin, et de droite à gauche.

Tableau d'analyse *SLR(1)* des expressions arithmétiques

	n	+	*	(	)	*$*	*E*	*T*	*F*
0	d5			d4			d1	d2	d3
1		d6				A			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			d8	d2	d3
5		r6	r6		r6	r6			
6	d5			d4				d9	d3
7	d5			d4					d10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Exemple : (3 + 2) * 4 *$*

Pile	Mot restant	Règle
0	(3 + 2) * 4 *$*	
0 4	3 + 2) * 4 *$*	
0 4 5	+ 2) * 4 *$*	
0 4 3	+ 2) * 4 *$*	$F \rightarrow$ **n**
0 4 2	+ 2) * 4 *$*	$T \rightarrow F$
0 4 8	+ 2) * 4 *$*	$E \rightarrow T$
0 4 8 6	2) * 4 *$*	
0 4 8 6 5	) * 4 *$*	
0 4 8 6 3	) * 4 *$*	$F \rightarrow$ **n**
0 4 8 6 9	) * 4 *$*	$T \rightarrow F$
0 4 8	) * 4 *$*	$E \rightarrow E + T$
0 4 8 11	* 4 *$*	
0 3	* 4 *$*	$F \rightarrow (E)$
0 2	* 4 *$*	$T \rightarrow F$
0 2 7	4 *$*	
0 2 7 5	*$*	
0 2 7 10	*$*	$F \rightarrow$ **n**
0 2	*$*	$T \rightarrow T * F$
0 1	*$*	$E \rightarrow T$
		ACCEPTÉ

Dérivation obtenue :

$E \Rightarrow T \Rightarrow T * F \Rightarrow T * 4 \Rightarrow F * 4$
$\Rightarrow (E) * 4 \Rightarrow (E + T) * 4$
$\Rightarrow (E + F) * 4 \Rightarrow (E + 2) * 4$
$\Rightarrow (T + 2) * 4 \Rightarrow (F + 2) * 4$
$\Rightarrow (3 + 2) * 4$

Parfois, les conflits ne peuvent pas être levés par l'examen des ensembles SUIVANT. On tient alors compte des caractères pouvant suivre une variable **pendant** la construction. Si l'automate ainsi construit n'a plus de conflits, on a une grammaire ***LR(1)***.

Inconvénient : très grand nombre d'états.

Solution de compromis : directement sur l'automate simple précédent, étudier la propagation des caractères pouvant suivre une variable. Dans le cas des réductions, cet ensemble joue le rôle des ensembles SUIVANT du cas *SLR(1)*. Si les conflits sont ainsi réglés, la grammaire est ***LALR(1)***.

Méthode adoptée par YACC.

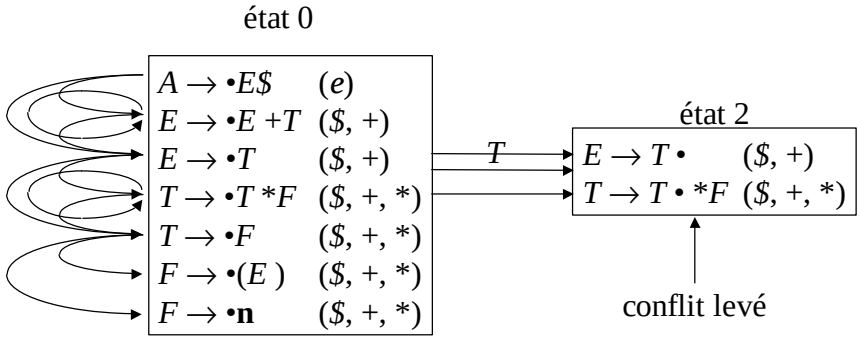
$$LR(1) \supset LALR(1) \supset SLR(1)$$

Règle de propagation du contexte :

item $x \rightarrow u \bullet yv \quad (z)$ donnant par clôture $y \rightarrow \bullet w \quad (\text{PREMIER}(vz))$

par transition $x \rightarrow uy \bullet v \quad (z)$

On construit le graphe de dépendance des items. On établit alors le contexte en suivant les flèches. Il peut y avoir des boucles.



Dans l'exemple, le résultat est le même que par *SLR(1)*.

Utilisation de l'ambiguïté :

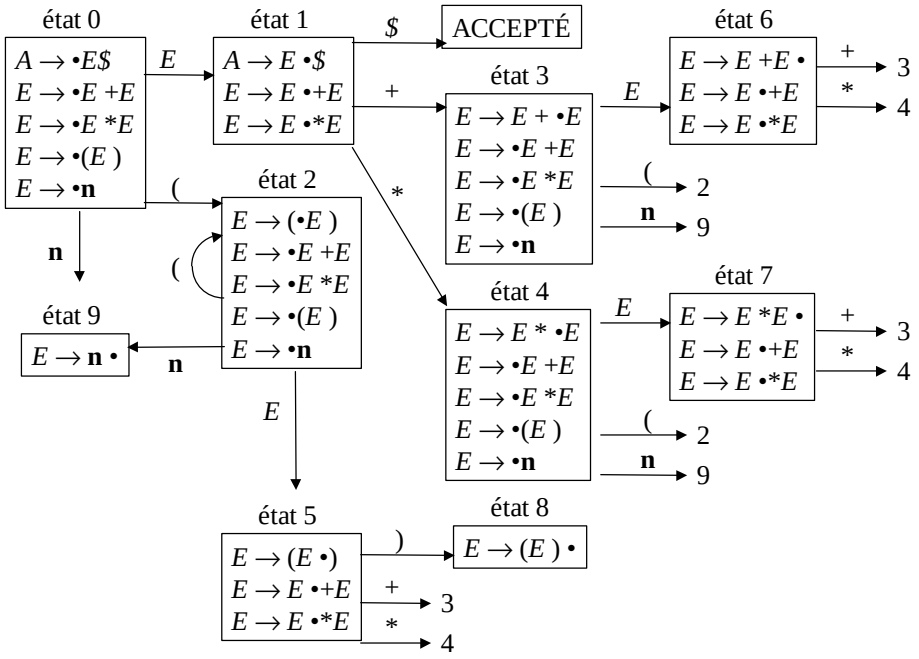
0 $A \rightarrow E\$$
1 $E \rightarrow E + E$
2 $E \rightarrow E * E$
3 $E \rightarrow (E)$
4 $E \rightarrow$ **n**

Grammaire ambiguë pour expressions arithmétiques

SUIVANT(E) = {$, +, *,)}

Ne peut être analysée par un analyseur déterministe, car ambiguë.

Néanmoins, des règles de priorité permettent de lever les conflits.



Conflits décalage-réduction

état 6

$E \rightarrow E + E \bullet$
$E \rightarrow E \bullet + E$
$E \rightarrow E \bullet * E$

Sur *$*,), pas de conflit, réduire
Sur +, réduire car + associatif à gauche
Sur *, décaler car * prioritaire sur +

état 7

$E \rightarrow E * E \bullet$
$E \rightarrow E \bullet + E$
$E \rightarrow E \bullet * E$

Sur *$*,), pas de conflit, réduire
Sur +, réduire car * prioritaire sur +
Sur *, réduire car * associatif à gauche

Si conflit entre $x \rightarrow u$ **op1**$y \bullet$ et $x \rightarrow u \bullet$**op2**y :

1) **op1** prioritaire sur **op2**, réduire ;
2) **op2** prioritaire sur **op1**, décaler ;
3) **op1** et **op2** ont même priorité :
   3.1) associativité à gauche, réduire ;
   3.2) associativité à droite, décaler ;
   3.3) pas d'associativité, signaler une erreur.

Récupération sur erreurs : Grammaire ambiguë précédente
Exemple de correction. Entrées corrigées en *italique*.

	n	+	*	(	)	*$*	*E*
0	d9	*1*	*1*	d2	*2*	*1*	d1
1	*3*	d3	d4	*3*	*2*	A	
2	d9	*1*	*1*	d2	*2*	*1*	d5
3	d9	*1*	*1*	d2	*2*	*1*	d6
4	d9	*1*	*1*	d2	*2*	*1*	d7
5	*3*	d3	d4	*3*	d8	*4*	
6	*r1*	r1	d4	*r1*	r1	r1	
7	*r2*	r2	r2	*r2*	r2	r2	
8	*r3*	r3	r3	*r3*	r3	r3	
9	*r4*	r4	r4	*r4*	r4	r4	

1 Empiler 9 (un **n** imaginaire) et signaler “opérande manquant”
2 Enlever) de l’entrée et signaler “parenthèse fermante en trop”
3 Empiler 3 (un + imaginaire) et signaler “opérateur manquant”
4 Empiler 8 et signaler “parenthèse fermante manquante”
Les réductions sur erreur repoussent la détection de l’erreur.