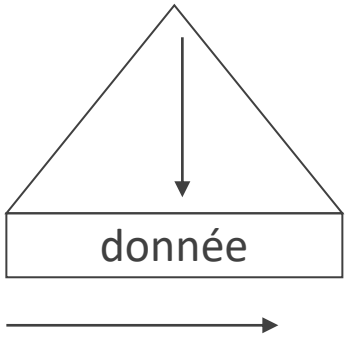
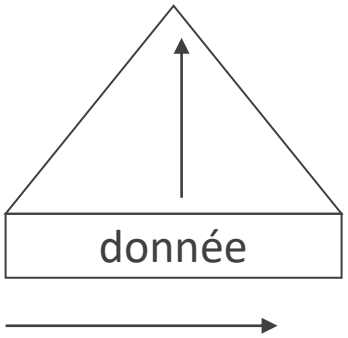
Analyse syntaxique
Les fonctions Premier et Suivant

Analyse syntaxique





Objectif

Construire l'arbre de dérivation d'une séquence donnée

Méthodes

Analyse descendante (*top-down parsing*) : on parcourt la séquence en appelant des fonctions pour chaque non-terminal

Analyse ascendante : on parcourt la séquence en empilant les symboles identifiés

Analyse descendante

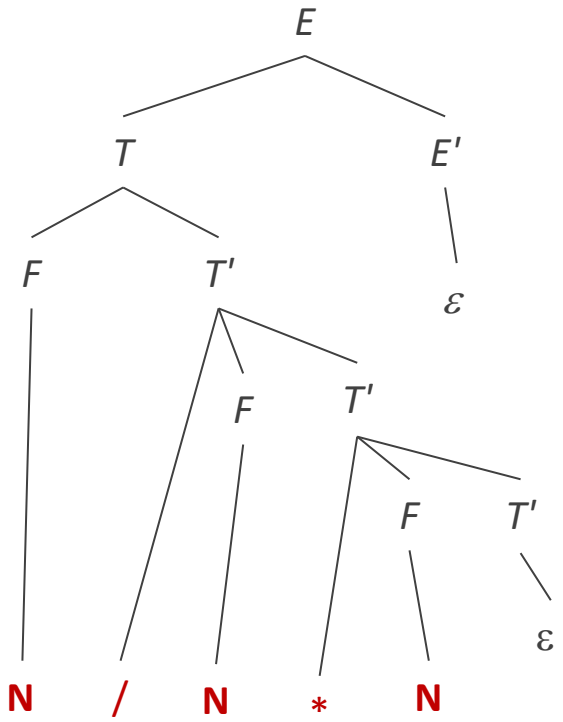
$E \rightarrow T\ E'$

$E' \rightarrow \mathbf{+}\ T\ E' \mid \mathbf{-}\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow \mathbf{*}\ F\ T' \mid \mathbf{/}\ F\ T' \mid \varepsilon$

$F \rightarrow \mathbf{(}\ E\ \mathbf{)} \mid \mathbf{N}$



Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Analyse descendante récursive

```
void expr(void) {
   int t;
   term();
   while(1)
     switch (next_token) {
     case '+': case '-':
       t = next_token;
       match(next_token); term(); emit(t, NONE);
       continue;
     default:
       return;
     }
}
```

- Exemple : traduction des expressions arithmétiques dans le mini-compilateur
- Incompatible avec les grammaires récursives à gauche
- Utilisée par gcc pour C++ depuis 2004 et pour C depuis 2006

```
void term(void) {
     int t;
     factor();
     while(1)
       switch(next_token) {
       case '*': case '/': case DIV: case MOD:
         t = next_token;
         match(next_token); factor(); emit(t, NONE);
         continue;
       default:
         return;
       }
}

void factor(void) {
     switch(next_token) {
         case '(':
            match('(');expr();match(')');break;
         case NUM:
            emit(NUM, tokenval); match(NUM); break;
         case ID:
            emit(ID, tokenval); match(ID); break;
         default:
            error("syntax error");
     }
}
```

Analyse descendante récursive

E --> $T\ E'$

E' --> **+** $T\ E'$ | **-** $T\ E'$ | ε

T --> $F\ T'$

T' --> ***** $F\ T'$ | **/** $F\ T'$ | ε

F --> **(** E **)** | **N**

Cette grammaire correspond à peu près au programme précédent

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Prédire quelle règle appliquer

```
void factor (void)
{
     switch(next_token) {
        case '(' :
           match('('); expr(); match(')'); break;
        case NUM :
           emit(NUM, tokenval); match(NUM); break;
        default:
           error("syntax error");
     }
}
```

	+	*	(	)	N	$
E			1		1	
E′	2			3		3
T			4		4	
T′	6	5		6		6
F			7		8	

1	*E* -->	*T E′*
2	*E′* -->	**+** *T E′*
3	*E′* -->	ε
4	*T* -->	*F T′*
5	*T′* -->	***** *F T′*
6	*T′* -->	ε
7	*F* -->	**(** *E* **)**
8	*F* -->	**N**

Prédire quelle règle appliquer

```
void factor(void)
{
     switch(next_token) {
         case '(':
            match('('); expr(); match(')'); break;
         case NUM:
            emit(NUM, tokenval); match(NUM); break;
         case ID:
            emit(ID, tokenval); match(ID); break;
         default:
            error("syntax error");
     }
}
```

	...	*a*	...
X		*r*	

Technique LL(1)

On prédit quelle règle appliquer en utilisant 1 lexème d'avance

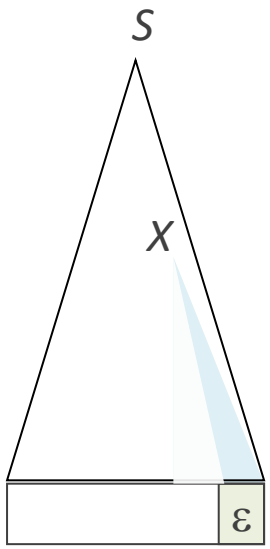
Il faut qu'il y ait au plus une règle par case du tableau

Les informations qu'on va utiliser sont :
- *X*, le non-terminal dans le nœud courant
- *a*, le prochain lexème

Selon ce tableau, il faut appliquer la règle *X* --> *r*

Un terminal spécial pour la fin

	...	*a*	...
X		ε	



E --> *T E′*
E′ --> **+** *T E′* | **-** *T E′* | ε
T --> *F T′*
T′ --> ***** *F T′* | **/** *F T′* | ε
F --> **(** *E* **)** | **N**

S --> *E* **$**
E --> *T E′*
E′ --> **+** *T E′* | **-** *T E′* | ε
T --> *F T′*
T′ --> ***** *F T′* | **/** *F T′* | ε
F --> **(** *E* **)** | **N**

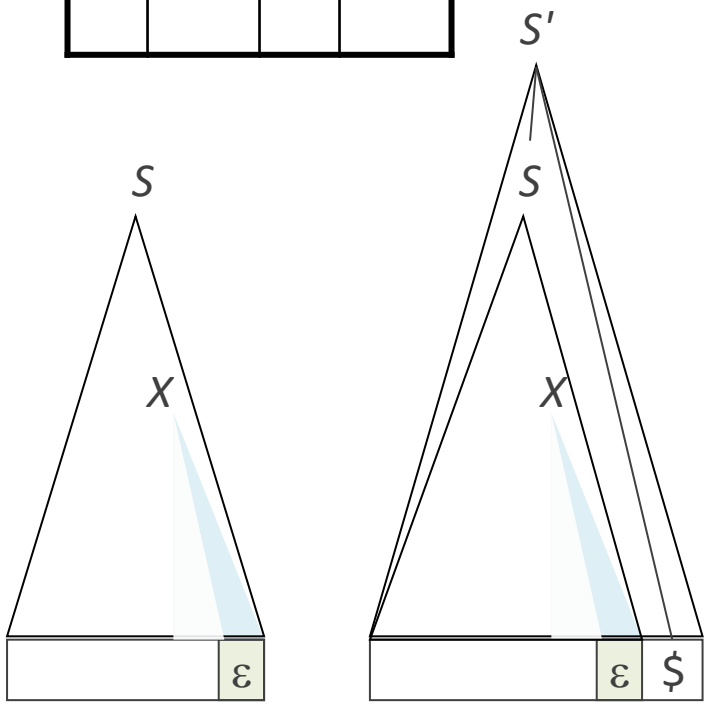
Que vaut *a* dans ce cas ?

On veut avoir un lexème d'avance même à la fin

Pour que *a* existe dans tous les cas, on veut qu'il existe un terminal $ qui n'apparaisse qu'à la fin de règles de la forme *S* --> *u* $ où *S* est l'axiome

Un terminal spécial pour la fin

	...	*a*	...
X		ε	

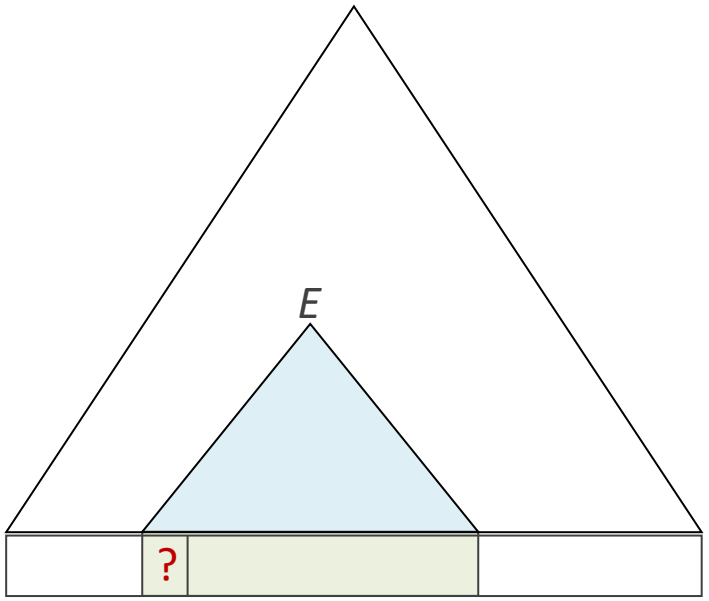


On veut qu'il existe un terminal $ qui n'apparaisse qu'à la fin de règles de la forme *S* --> *u* $ où *S* est l'axiome

S'il n'y en a pas, on change d'axiome et on ajoute une règle *S'* --> *S* $

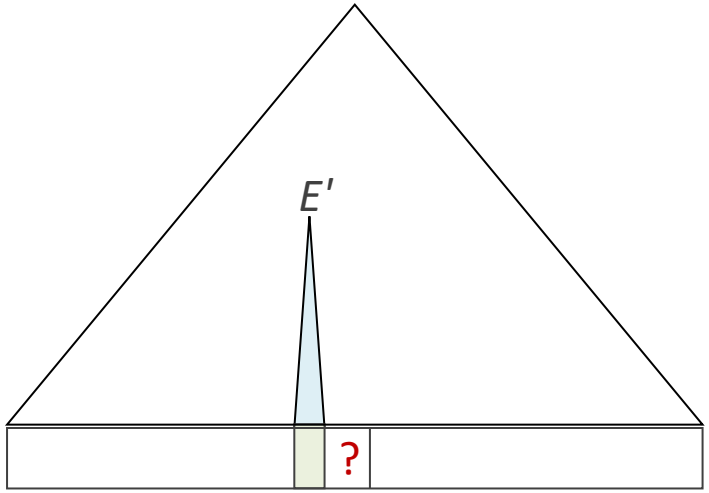
a vaut $

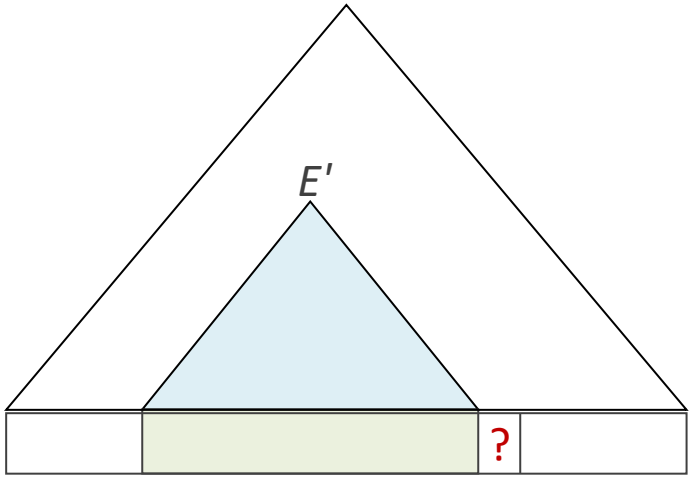
Prédiction



1	*E* -->	*T E′*
2	*E′* -->	**+** *T E′*
3	*E′* -->	ε
4	*T* -->	*F T′*
5	*T′* -->	***** *F T′*
6	*T′* -->	ε
7	*F* -->	**(** *E* **)**
8	*F* -->	**N**

Étant donné un *E* dans un arbre de dérivation, par quels terminaux peut commencer la frontière du sous-arbre dominé par *E* ?





Prédiction

0	*S* -->	*E* $
1	*E* -->	*T E′*
2	*E′* -->	**+** *T E′*
3	*E′* -->	ε
4	*T* -->	*F T′*
5	*T′* -->	***** *F T′*
6	*T′* -->	ε
7	*F* -->	**(** *E* **)**
8	*F* -->	**N**

Étant donné un *E′* dans un arbre de dérivation, quels terminaux peuvent suivre la frontière du sous-arbre dominé par *E′* ?

Prédiction

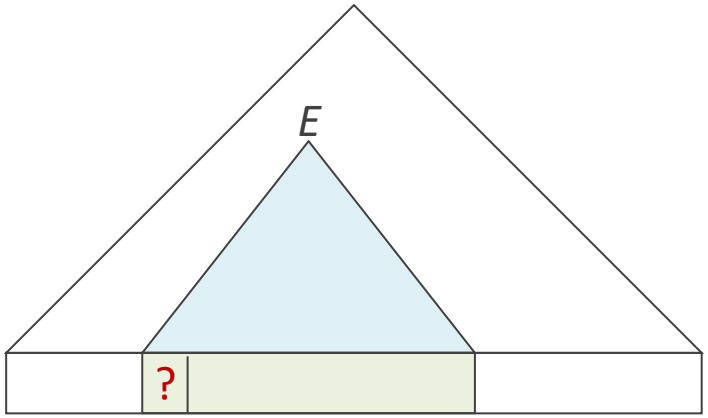
$$
\begin{cases}
E \rightarrow T\ E' \\
E' \rightarrow \mathbf{+}\ T\ E' \mid \mathbf{-}\ T\ E' \mid \varepsilon \\
T \rightarrow F\ T' \\
T' \rightarrow \mathbf{*}\ F\ T' \mid \mathbf{/}\ F\ T' \mid \varepsilon \\
F \rightarrow \mathbf{(}\ E\ \mathbf{)} \mid \mathbf{N}
\end{cases}
$$

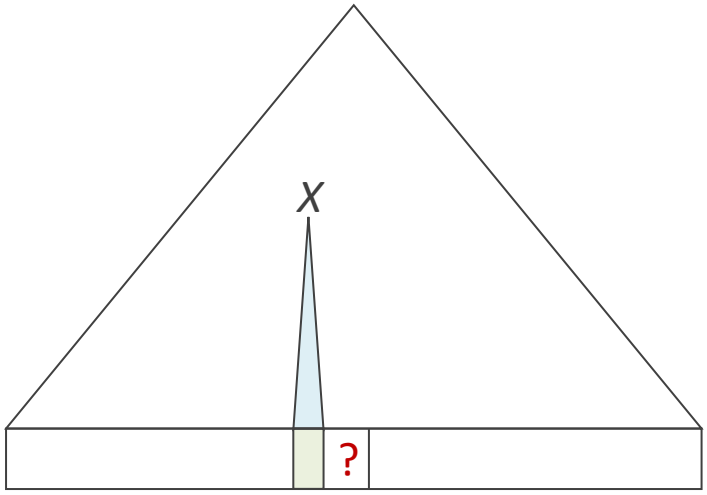
Comment être sur de

- prédire correctement la règle à appliquer ?
- détecter qu'aucune règle n'est applicable ?

Automatiser la construction du tableau LL(1)

Premier et Suivant





On définit deux fonctions

Premier() de $(A \mid V)^*$ dans $\mathcal{P}(A \cup \{\varepsilon\})$

$a \in A$: a est dans Premier(u) ssi on peut dériver à partir de u un mot commençant par a

ε est dans Premier(u) ssi on peut dériver ε à partir de u

Suivant() de V dans $\mathcal{P}(A)$

$a \in A$: a est dans Suivant(X) ssi on peut dériver à partir de l'axiome un mot dans lequel un X est suivi d'un a

Premier et Suivant

E --> *E* **+** *T* | *E* **-** *T* | *T*

T --> *T* ***** *F* | *T* **/** *F* | *F*

F --> **(** *E* **)** | **N**

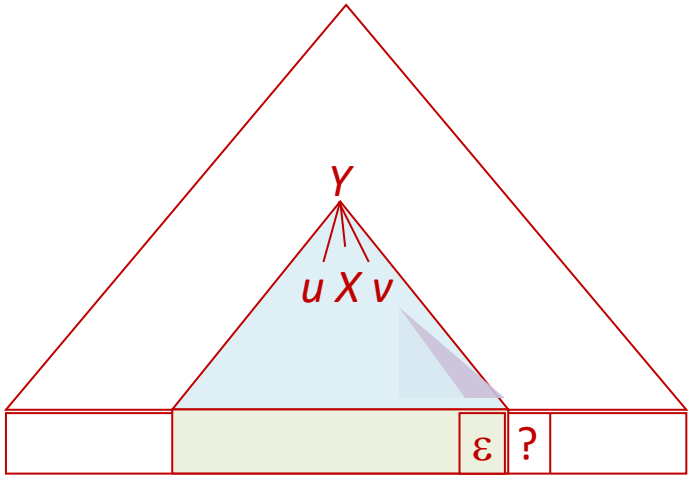
Premier()

(est dans Premier(*T*) car *T* --> *F* --> **(** *E* **)**

Suivant()

) est dans Suivant(*T*) car *E* --> *T* --> *F* --> **(** *E* **)** --> **(** *E* **+** *T* **)**

Savoir calculer Premier() et Suivant() servira aussi pour utiliser Bison (analyse ascendante)



Calcul de Premier et Suivant

Algorithme par graphes

On construit deux graphes dont les sommets sont des symboles et ε

On aura un **chemin** de X à $a \in A$ ssi a est dans Premier(X) (ou Suivant(X))

On aura un **arc** de X à ε ssi ε est dans Premier(X)

Premier()

On a un arc de X vers $Y \in A \cup V$ ssi il existe une règle $X \dashrightarrow u\ Y\ v$ avec $u \overset{*}{\dashrightarrow} \varepsilon$

On a un arc de X vers ε ssi $X \overset{*}{\dashrightarrow} \varepsilon$

Suivant()

On a un arc de X vers $a \in A$ ssi il existe une règle $Y \dashrightarrow u\ X\ v$ avec $a \in$ Premier(v)

On a un arc de X vers Y ssi il existe une règle $Y \dashrightarrow u\ X\ v$ avec $v \overset{*}{\dashrightarrow} \varepsilon$

(attention, l'arc remonte la flèche de dérivation, car Suivant(Y) $\subseteq$ Suivant(X))

Exemple

S --> E **$**

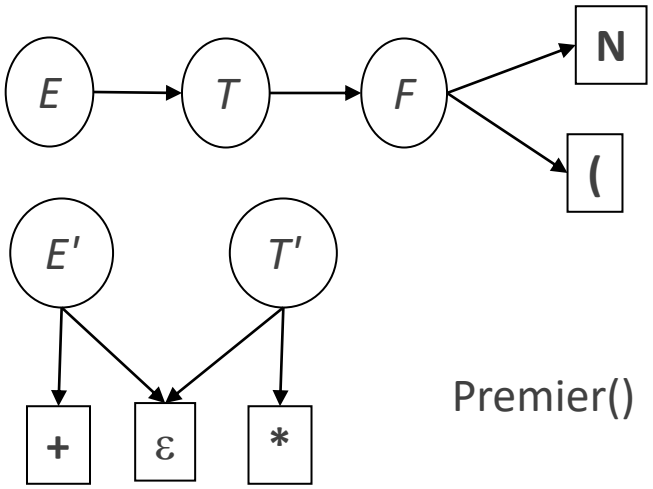
E --> $T\ E'$

E' --> **+** $T\ E'$ | ε

T --> $F\ T'$

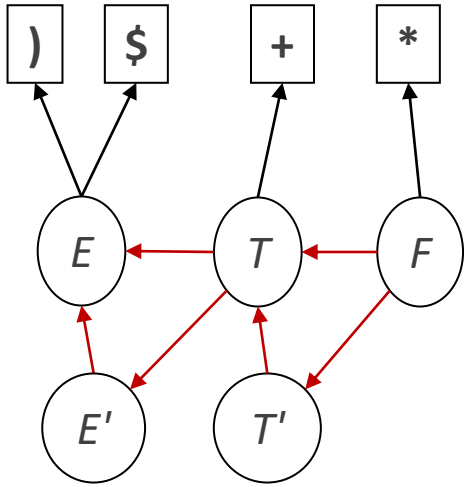
T' --> ***** $F\ T'$ | ε

F --> **(** E **)** | **N**



Premier()

Suivant()



Calcul de Premier et Suivant

$$
\begin{cases}
S \dashrightarrow E\ \$ \\
E \dashrightarrow T\ E' \\
E' \dashrightarrow +\ T\ E' \mid \varepsilon \\
T \dashrightarrow F\ T' \\
T' \dashrightarrow *\ F\ T' \mid \varepsilon \\
F \dashrightarrow (\ E\ ) \mid \mathbf{N}
\end{cases}
$$

Étape	1	2	3	4.1	4.2
E	ø	ø	ø	ø	{(, **N**}
E'	ø	{ε}	{ε, **+**}	{ε, **+**}	{ε, **+**}
T	ø	ø	ø	{(, **N**}	{(, **N**}
T'	ø	{ε}	{ε, ***** }	{ε, ***** }	{ε, ***** }
F	ø	ø	{(, **N**}	{(, **N**}	{(, **N**}

Algorithme par tableaux

On construit deux tableaux avec une ligne pour chaque non-terminal

Premier()

1. Initialiser chaque Premier(*X*) à l'ensemble vide
2. Ajouter ε à Premier(*X*) ssi *X* --> ε
3. Ajouter $a \in A$ ssi il existe une règle *X* --> *a u* avec $u \in (A \mid V)^*$
4. Ajouter Premier(*Y*) ssi il existe une règle *X* --> *u Y v* avec *u* $\overset{*}{\dashrightarrow}$ ε. Itérer l'étape 4 tant qu'un des ensembles change

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Calcul de Premier et Suivant

$$
\begin{cases}
S \dashrightarrow E\ \$ \\
E \dashrightarrow T\ E' \\
E' \dashrightarrow +\ T\ E' \mid \varepsilon \\
T \dashrightarrow F\ T' \\
T' \dashrightarrow *\ F\ T' \mid \varepsilon \\
F \dashrightarrow (\ E\ ) \mid \mathbf{N}
\end{cases}
$$

Étape	1	2	3.1	3.2
E	ø	{$,)}	{$,)}	{$,)}
E'	ø	ø	{$,)}	{$,)}
T	ø	{+}	{+, $,)}	{+, $,)}
T'	ø	ø	{+}	{+, $,)}
F	ø	{*}	{*, +, $,)}	{*, +, $,)}

Suivant()

1. Initialiser chaque Suivant(*X*) à l'ensemble vide
2. Pour chaque une règle *Y* --> *u X v*, ajouter Premier(*v*) \ {ε}
3. Pour chaque une règle *Y* --> *u X v* telle que *v* $\overset{*}{\dashrightarrow}$ ε, ajouter Suivant(*Y*)

Itérer l'étape 3 tant qu'un des ensembles change