

## DONNÉES POUR EXPLICATION :

on utilisera ça :

$$D \rightarrow T \text{ ID } (' L ') ' ;$$

$$L \rightarrow P \text{ R} \mid \epsilon$$

$$R \rightarrow ',' \text{ P R} \mid \epsilon$$

$$P \rightarrow T \text{ ID}$$

$$T \rightarrow \text{INT} \mid \text{FLOAT}$$

### Calculer Premier ?

On prend le **premier élément** de chaque OU ('|') de la partie droite de la règle, si c'est un terme terminal on prend ça, sinon, si c'est un non-terminal, on prend les premiers de ce non terminal. /!\ il me semble que si ce non terminal est ANNULABLE (c'est à dire qu'il est possible qu'il rende  $\epsilon$ ), on prends les premiers de ce non terminal ET ceux du terme suivant, si lui aussi est annulable, on prend les premiers et on passe au suivant, etc.. jusqu'à arriver à un non annulable ou plus de termes disponibles.

$$\text{Premiers}(T) = \{\text{INT}, \text{FLOAT}\}$$

$$\text{Premiers}(P) = \text{Premiers}(T) = \{\text{INT}, \text{FLOAT}\}$$

$$\text{Premiers}(R) = \{',', \epsilon\}$$

$$\text{Premiers}(L) = \text{Premiers}(P) \cup \{\epsilon\} = \{\text{INT}, \text{FLOAT}, \epsilon\}$$

$$\text{Premiers}(D) = \text{Premiers}(T) = \{\text{INT}, \text{FLOAT}\}$$

### Calculer Suivants ?

Comment trouver les suivants : il y a une **phase d'initialisation** et une **phase de propagation**.  
Ce qu'il faut faire :

Phase d'initialisation : chercher une occurrence de la règle étudiée parmi les parties droites des règles. Ensuite, on regarde ce qu'il y a juste après. Si ce qui vient juste après est non terminal,

Suivant (D) : on cherche D dans les parties droites des règles. Il n'y est pas (car c'est l'axiome), donc les suivants de D c'est  $\$$  (car c'est l'axiome).

Suivant(L) : on cherche L dans les parties droite des règles. Il apparaît uniquement dans cette règle :

$$D \rightarrow T \text{ ID } (' L ') ' ;$$

Ce qui vient juste après c'est  $'$ . Comme c'est terminal, c'est notre suivant de L.

suivant(P) = R. Comme c'est non terminal, on regarde les premiers de rparam. suivant(P) = premier(R)

Phase de propagation : on regarde la fin de chaque règle, et on ajoute les suivant de la partie de gauche de la flèche aux suivant du dernier paramètre des trucs à droite de la flèche. On applique la règle de propagation jusqu'à ce que ça ne change plus (AJOUTER les suivants d'une règle à une autre correspond à faire l'union des deux ensembles de suivants).

propagation par la règle :  $L \rightarrow P R | \epsilon$

- on ajoute les suivant (L) aux suivant(R)
- comme R est annulable, on ajoute les suivant(L) aux suivant(P)

SUIVANT	D	L	R	P	T
init	\$	)	none	,	ID
propagation	\$	)	)	,)	ID
propagation2 (inutile)	\$	)	)	,)	ID

## $\epsilon$ production ?

Soit la grammaire  $G = (\{a, b\}, \{S, X, Y\}, R, S)$

avec les règles de R suivantes :

$S \rightarrow aX|Y|XX$

$X \rightarrow \epsilon|b|XX$

$Y \rightarrow aXb$

On calcule les ensembles d'effaçables :  $E1 = \{X\}$ ,  $E2 = \{X, S\}$ ,  $E3 = \{X, S\}$ .

On obtient donc un nouvel ensemble de règles

R1 :

$S \rightarrow aX|a|Y|XX|X$

$X \rightarrow b|XX|X$

$Y \rightarrow aXb|ab$

## Collection Canonique SLR ?

On crée une nouvelle règle, ici  $S \rightarrow D \$$ , qui n'est jamais à droite de la flèche et qui sert de point de départ, D étant la règle "point de départ" de notre syntaxe. Ensuite on crée notre premier état,  $E0$ , on note tout en haut  $S \rightarrow .D \$$  (le point permet d'indiquer où on en est de notre analyse. comme on rencontre un D après le point, en dessous de  $S \rightarrow .D \$$ , on note toutes les possibilités d'analyse de D, donc ici  $D \rightarrow .T ID ( L )$ , on note alors en dessous toutes les possibilités d'analyse de T, ici  $T \rightarrow .INT$  et  $T \rightarrow .FLOAT$ .

Première étape finie, ensuite on va créer des nouveaux états en fonction du terme qui est lu:

Un premier état  $E1$  si ce qui est lu est un D, on rentre dans le cas  $S \rightarrow D \$$ , qui n'a aucune sous analyse (à noter que le point se déplace afin d'indiquer où on en est de l'analyse), on

fait pareil pour T, INT et FLOAT, puis à nouveau pour ID, et c'est ici que ça devient intéressant, en lisant une parenthèse ouvrante, on crée un état tel qu'on en est à :  
 $D \rightarrow T \text{ ID} ( .L )$

Ici on lit un L, on ajoute alors toutes les possibilités de lecture d'un L en dessous :

$L \rightarrow .$  (cas de retour epsilon)

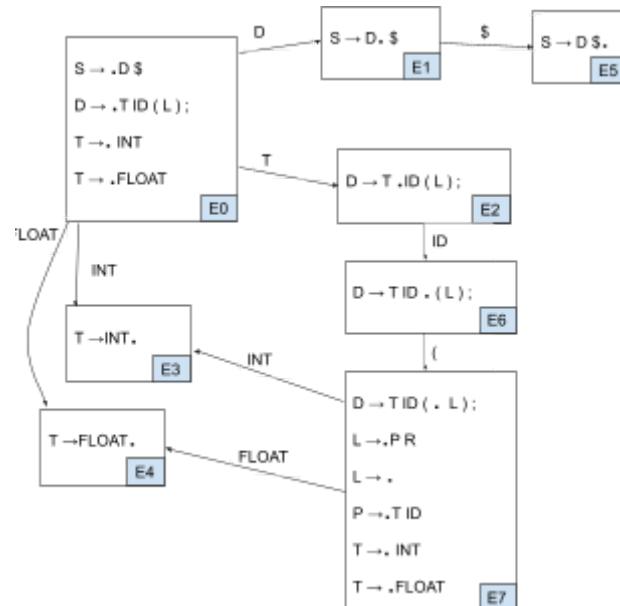
$L \rightarrow .P R$

vu qu'on lit un P, on ajoute toutes les possibilités du P:

$P \rightarrow .T \text{ ID}$

on a un T donc on ajoute :

$T \rightarrow .INT$  et  $T \rightarrow .FLOAT$



## Table d'analyse ascendante ?

On crée un tableau avec comme nombre de ligne le nombre d'état, on divise les colonnes en deux types, les terminaux (action) et non terminaux (successeurs), pour chaque état, on note dans la case ce qu'il se passe lorsqu'on rencontre le terme indiqué dans la colonne, si l'état est un état final on note dans la case du terme qui amène à cet état "R(règle de l'état final)". Sinon on indique simplement (si dans la partie ACTION) Shift 'état d'arrivé' ou (si dans la partie SUCCESSEURS) simplement 'état d'arrivé'. On aura donc une table LR(0) qui peut avoir des conflits.

Dans le cas où dans une seule case on a un shift (S) et un reduce (R) (conflit), on peut créer une table d'analyse SLR(1). Imaginons que l'on ait par exemple  $R(e \rightarrow L)$  (avec e et L arbitraire), on regarde les suivants de e (la partie gauche de la flèche) et on ne gardera le reduce que dans les cases des termes appartenants aux suivants de e. On fait ça pour toute la table, pas que pour les zones de conflits.

S'il reste des conflits, cela veut dire que la grammaire est ambiguë, dans le cadre d'une programmation bison il faut préciser dans ses règles quelle est la priorité, par défaut bison priorise le Shift.

Quelques petits exemples ci dessous:

	ACTION					SUCCESEURS			
	\$	(	)	INT	FLOAT	T	L	D	P
E0				S -> E3		E2			
E4					R(T->FL OAT)				

Table d'analyse descendante ?

symboles non terminaux : D, L, R, P, T:

les lignes correspondent aux règles, et les colonnes correspondent à l'ensemble des terminaux + \$

Dans un premier temps : on regarde les premiers de chaque règles. Si je les croise, j'applique les règles correspondantes.

premier(D) = INT, FLOAT. Dans leur case, je met la règle  $D \rightarrow T \text{ ID } ('L')$  ;

Dans un second temps : on regarde les règles effaçables (de type :  $X \rightarrow \epsilon$ ). Je regarde les suivants de cette règle (la règle X), et j'applique la règle aux suivants.

par exemple : on a la règle  $L \rightarrow \epsilon$ . On regarde les suivants de L. suivant(L) = ')'. Donc, quand je croise un ) quand je suis dans L, j'applique la règle  $L \rightarrow \epsilon$

explication : quand je lit un mot, et qu'après mon L je trouve un ), c'est donc que j'ai utilisé la règle  $L \rightarrow \epsilon$

Un conflit signifie qu'il est possible de mettre deux dérivations dans une même case, cela signifie alors que notre grammaire n'est pas LL(1).

	,	ID	(	INT	FLOAT	)	;	\$
d				$D \rightarrow T \text{ ID } ('L')$ ;	$D \rightarrow T \text{ ID } ('L')$ ;			
l				$L \rightarrow P \text{ R}$	$L \rightarrow P \text{ R}$	$L \rightarrow \epsilon$		
r	$R \rightarrow P \text{ R}$					$R \rightarrow \epsilon$		
p				$P \rightarrow T \text{ ID }$	$P \rightarrow T \text{ ID }$			

t				T→INT	T→FLOAT			
---	--	--	--	-------	---------	--	--	--

## Représentation automate à pile descendant (BISON) ?

Chose à reconnaître : int f(float f,int x ); représenté dans la pile par D \$ (souligné = tête de lecture)

pile (sommet de pile à gauche)	Reste à lire	action
D \$	<u>int</u> f(float f,int x );\$	D→T ID (L) ;
I ID (L) ; \$	<u>int</u> f(float f,int x );\$	T→INT
INT ID (L) ; \$	<u>int</u> f(float f,int x );\$	AVANCER
ID (L) ; \$	<u>f</u> (float f,int x );\$	AVANCER
(L) ; \$	<u>(</u> float f,int x );\$	AVANCER
L) ; \$	<u>float</u> f,int x );\$	L→P R
P R) ; \$	<u>float</u> f,int x );\$	P→T ID
I ID R) ; \$	<u>float</u> f,int x );\$	T→FLOAT
FLOAT ID R) ; \$	<u>float</u> f,int x );\$	AVANCER
ID R) ; \$	<u>f</u> ,int x );\$	AVANCER
R) ; \$	<u>,</u> int x );\$	R→, P R
_ P R) ; \$	<u>_</u> int x );\$	AVANCER
P R) ; \$	<u>int</u> x );\$	P -> T ID
I ID R) ; \$	<u>int</u> x );\$	T → INT
INT ID R) ; \$	<u>int</u> x );\$	AVANCER
ID R) ; \$	<u>x</u> );\$	AVANCER
R) ; \$	<u>_</u> ; \$	R→ε
_ ; \$	<u>_</u> ; \$	AVANCER
_ ; \$	<u>_</u> \$	AVANCER
\$	\$	ACCEPTER

## Représentation automate à pile ascendant (BISON) ?

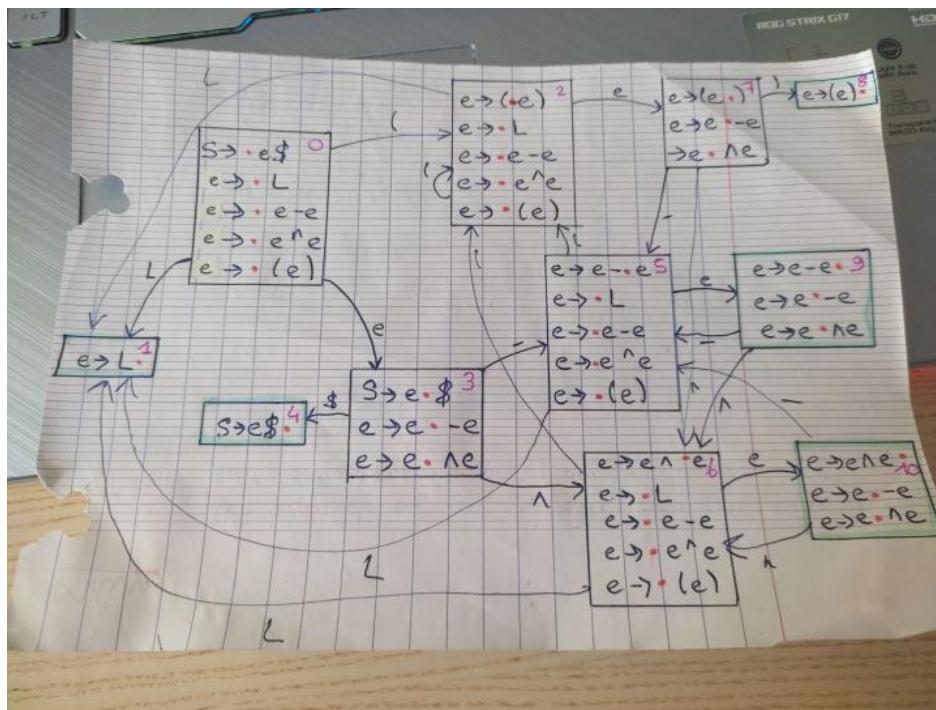
On prendra pour cet exemple la grammaire suivante :

$G = (VT = \{LIT F LOT, -, b, (, )\}, VN = \{\text{exp}\}, R, \text{exp})$

avec pour règles :

$\text{exp} \rightarrow \text{LITFLOT} \mid \text{exp} - \text{exp} \mid \text{exp} \wedge \text{exp} \mid (\text{exp})$

avec comme table SLR :



on a comme automate (et comme arbre de dérivation ) ce résultat :

pile		Flot	Acr°	term
[E0]		2^3 - 7.0	Shift E1	
[E0] → 2 [E1]		↑3 - 7.0	R(e→l)	2
[E0] → e [E3]		3 - 7.0	Shift E6	
[E0] → e [E3] → n [E6]		3 - 7.0	Shift E1	
[E0] → e [E3] → n [E6] → 3 [E1]		- 7.0	Reduce(e→l)	
[E0] → e [E3] → n [E6] → c [E10]		- 7.0	R(e-x'e)	
[E0] → e [E3]		- 7.0	S(ES)	
[E0] → e [E3] → - [E5]		- 7.0	S E1	
[E0] → e [E3] → - [E5] → 7.0 [E4]		\$	R(e→l)	
[E0] → e [E3] → - [E5] → e / [E9]		\$	R(e-de-e)	
[E0] → e [E3]		\$	S(E4)	
[E0] → & [E3] → & [E4]		\$	Accept	

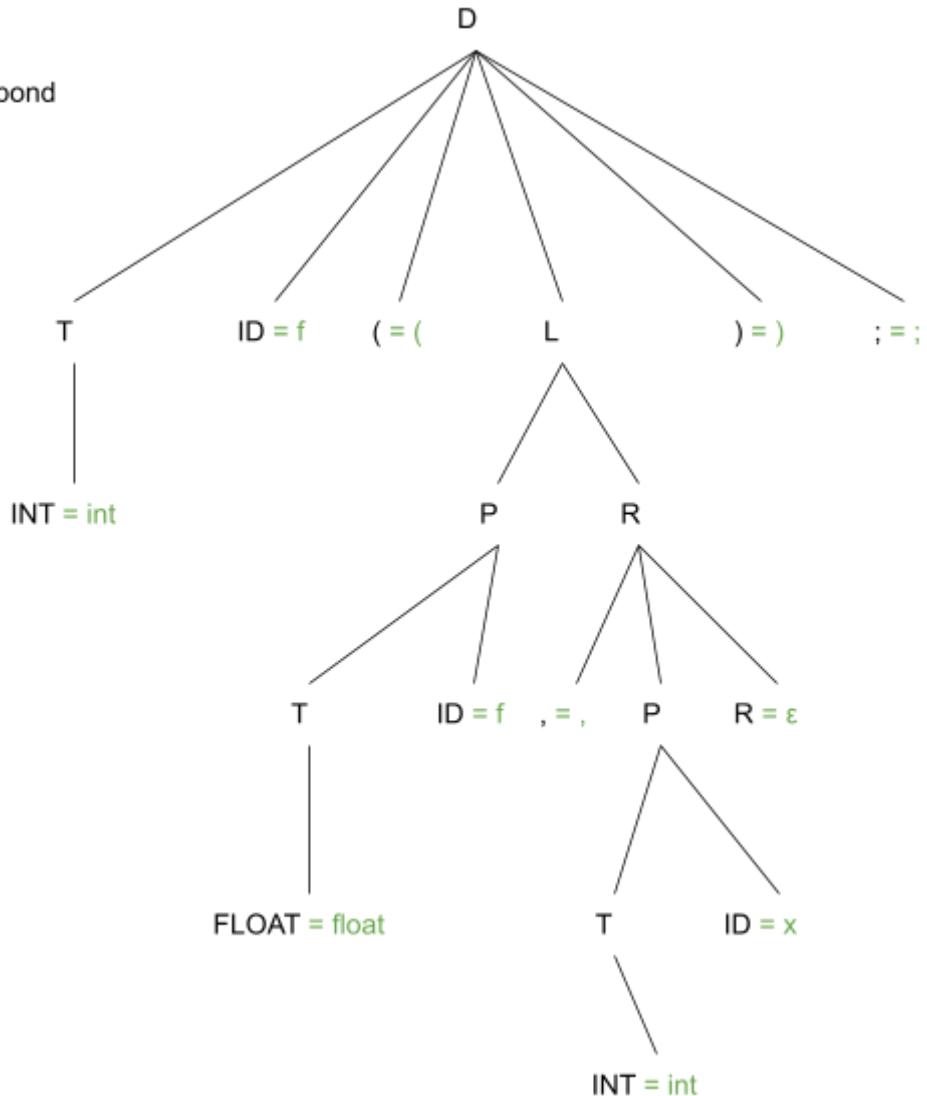
```

graph TD
    A[&] --> B[e]
    A --> C[e]
    B --> D[L]
    B --> E[L]
    C --> F[L]
    C --> G[L]
    D --> H[2]
    D --> I[3]
    E --> J[2]
    E --> K[3]
  
```

Arbre de dérivation ?

En utilisant la représentation précédente (automate à pile DESCENDANT):

texte en vert correspond  
à l'attribution de  
l'analyse.



A noter que si toutes les manipulations précédentes ne sont pas possibles (automate à pile, arbre, table d'analyse descendante), notre syntaxe n'est pas LL(1).

Définitions :

Une grammaire  $G = (VT, VN, R, S)$  est **ambiguë** si et seulement s'il existe deux dérivations gauches distinctes partant de  $S$  et aboutissant au même mot terminal  $m$ . (s'il existe 2 arbres de dérivation différents)

**l'analyse syntaxique descendante** consiste à partir de l'axiome qui constitue la racine de l'arbre de dérivation (ou arbre syntaxique). L'arbre de dérivation est ainsi construit (ou pas) depuis la racine  $S$  vers les feuilles.

Analyse descendante impossible sur les grammaires récursives à gauche. La récursivité à droite est autorisé

**l'analyse syntaxique ascendante** consiste, au contraire, à partir du programme et à remonter vers l'axiome  $S$ . L'arbre de dérivation est alors construit (ou pas) depuis les feuilles vers la racine  $S$