Introduction à la compilation
Cours 4 : Analyse syntaxique ascendante

Yann Régis-Gianas
yrg@pps.univ-paris-diderot.fr

PPS - Université Denis Diderot – Paris 7

Exemple d'analyse ascendante

- Soit la grammaire (tirée de *Parsing Techniques* de Grune et Jacobs) :

(1)	S	$\rightarrow$	aSQ
(2)	S	$\rightarrow$	abc
(3)	bQc	$\rightarrow$	$bbcc$
(4)	cQ	$\rightarrow$	Qc

et l'entrée : « *aabbcc* »

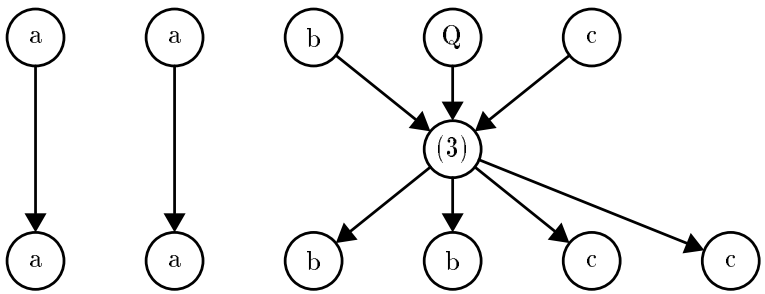
- Quelle règle peut produire une sous-séquence de non-terminaux de ce mot ?

Exemple d'analyse ascendante

- Soit la grammaire (tirée de *Parsing Techniques* de Grune et Jacobs) :

(1)	S	$\rightarrow$	aSQ
(2)	S	$\rightarrow$	abc
(3)	bQc	$\rightarrow$	$bbcc$
(4)	cQ	$\rightarrow$	Qc

et l'entrée : « *aabbcc* »



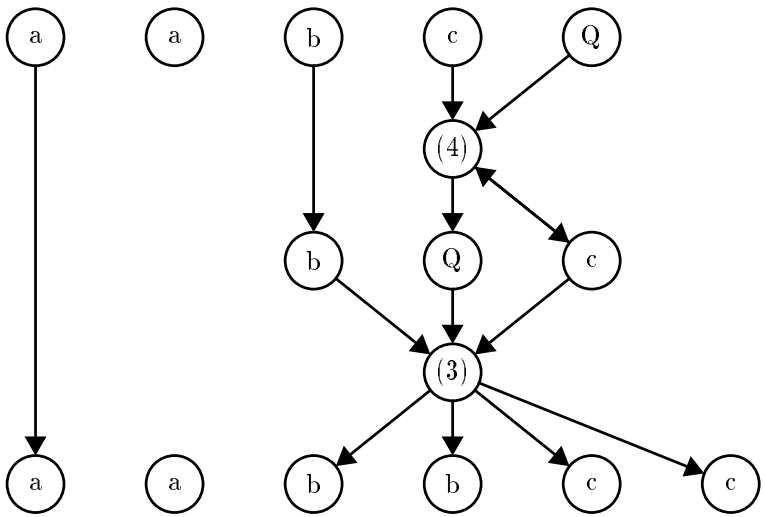
- De même, seule la règle (4) a pu produire le mot « Q c ».

Exemple d'analyse ascendante

- Soit la grammaire (tirée de *Parsing Techniques* de Grune et Jacobs) :

(1)	S	$\rightarrow$	aSQ
(2)	S	$\rightarrow$	abc
(3)	bQc	$\rightarrow$	$bbcc$
(4)	cQ	$\rightarrow$	Qc

et l'entrée : « *aabbcc* »



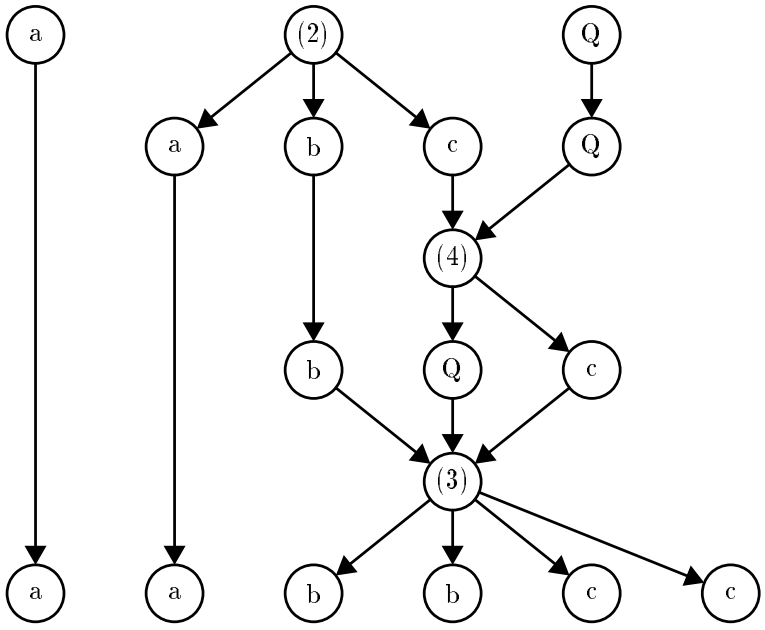
- Ici, seule la règle (3) s'applique et elle mène à l'entrée.

Exemple d'analyse ascendante

- Soit la grammaire (tirée de *Parsing Techniques* de Grune et Jacobs) :

(1)	S	$\rightarrow$	aSQ
(2)	S	$\rightarrow$	abc
(3)	bQc	$\rightarrow$	$bbcc$
(4)	cQ	$\rightarrow$	Qc

et l'entrée : « *aabbcc* »



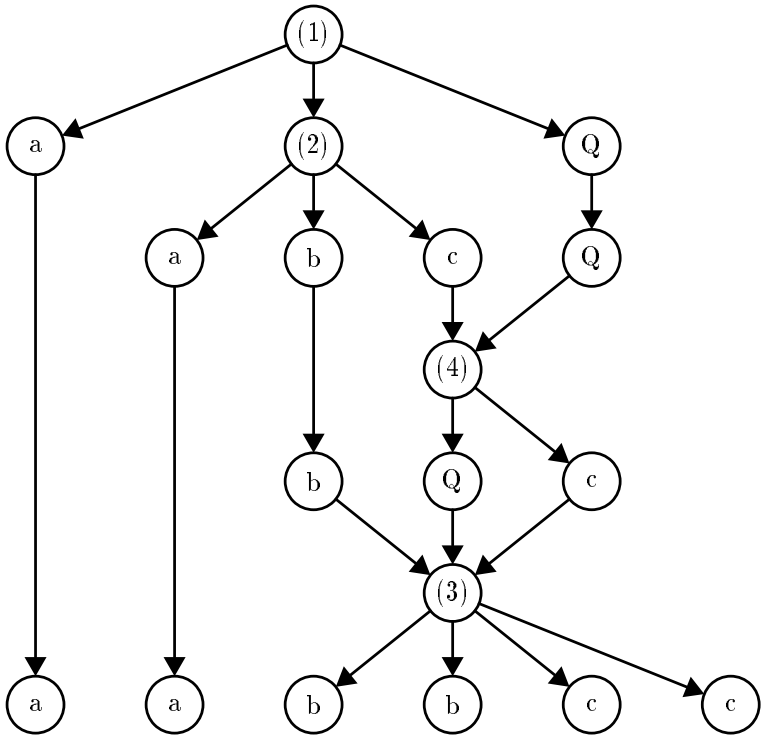
- Encore une fois, une seule règle s'applique, la règle (2).

Exemple d'analyse ascendante

- Soit la grammaire (tirée de *Parsing Techniques* de Grune et Jacobs) :

(1)	S	$\rightarrow$	aSQ
(2)	S	$\rightarrow$	abc
(3)	bQc	$\rightarrow$	$bbcc$
(4)	cQ	$\rightarrow$	Qc

et l'entrée : « *aabbcc* »



- Pour finir, seule la règle (1) produit le mot « aSQ ».

Algorithme CYK

Principe

- L'algorithme CYK [1] construit une table dont la dernière ligne contient le mot à produire et les lignes qui la précèdent contiennent les non-terminaux qui ont pu conduire à la production d'une certaine sous-chaîne :

S				
	C			
	B		B, C	
A				
a	a	b	b	c

Table construite pour la reconnaissance du mot *aabbc* par la grammaire :

$$
\begin{aligned}
S &::= AC \\
A &::= a \\
B &::= ab \mid bc \\
C &::= bc \mid BC
\end{aligned}
$$

- Cette table est construite de la dernière ligne à la première, ce qui confère un caractère ascendant à cette analyse syntaxique.

1. Du nom de ces inventeurs J. Cocke, D.H. Younger et T. Kasami.

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

- On se concentre sur les non-terminaux qui produisent des sous-chaînes de taille 1.

/	a	a	/	b	/	b	b	/

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

- On se concentre sur les non-terminaux qui produisent des sous-chaînes de taille 1.

	A	A		B		B	B	
$/$	a	a	$/$	b	$/$	b	b	$/$

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

- Il faut prendre en compte les règles unitaires.

	E, A	E, A		E, B		E, B	E, B	
$/$	a	a	$/$	b	$/$	b	b	$/$

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

- On peut passer aux non-terminaux produisant des chaînes de taille 2.

	E, A					E, B		
	E, A	E, A		E, B		E, B	E, B	
$/$	a	a	$/$	b	$/$	b	b	$/$

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$\begin{array}{lcl} S & ::= & / \; L \\ L & ::= & \epsilon \mid E \; / \; L \\ E & ::= & A \mid B \\ A & ::= & aa \mid a \\ B & ::= & bb \mid b \end{array}$$

- Pour reconnaître la fin de la chaîne comme une production de L, il faut « insérer » un mot vide à la fin de la chaîne.

$\Rightarrow$ En fait, il faudrait le faire entre tous les terminaux et au début de la chaîne.

						L			
	E, A					E, B			
	E, A	E, A		E, B		E, B	E, B		
$/$	a	a	$/$	b	$/$	b	b	$/$	ϵ

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

- On continue l'analyse en s'intéressant à des chaînes de plus en plus longues.

<table>
<tr><td></td><td colspan="9">L</td></tr>
<tr><td colspan="3"></td><td colspan="7">L</td></tr>
<tr><td colspan="5"></td><td colspan="5">L</td></tr>
<tr><td></td><td colspan="3">E, A</td><td></td><td></td><td colspan="3">E, B</td><td></td><td></td></tr>
<tr><td></td><td colspan="2">E, A</td><td>E, A</td><td></td><td>E, B</td><td></td><td>E, B</td><td>E, B</td><td></td><td></td></tr>
<tr><td>$/$</td><td colspan="2">a</td><td>a</td><td>$/$</td><td>b</td><td>$/$</td><td>b</td><td>b</td><td>$/$</td><td>ϵ</td></tr>
</table>

Exemple de dérivation détaillée

- Analysons le mot « $/aa/b/bb/$ » à l'aide de la grammaire G :

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

- L'analyse se termine lorsque l'on a reconnu l'ensemble du mot à analyser comme une production du symbole d'entrée de la grammaire.

<table>
<tr><td colspan="13">S</td></tr>
<tr><td></td><td colspan="12">L</td></tr>
<tr><td colspan="5"></td><td colspan="8">L</td></tr>
<tr><td colspan="9"></td><td colspan="4">L</td></tr>
<tr><td></td><td colspan="4">E, A</td><td colspan="2"></td><td></td><td colspan="3">E, B</td><td></td><td></td></tr>
<tr><td></td><td colspan="2">E, A</td><td>E, A</td><td></td><td colspan="2">E, B</td><td></td><td>E, B</td><td>E, B</td><td></td><td></td></tr>
<tr><td>$/$</td><td>a</td><td></td><td>a</td><td>$/$</td><td colspan="2">b</td><td>$/$</td><td>b</td><td>b</td><td>$/$</td><td>ϵ</td></tr>
</table>

Description formelle de l'algorithme (hors programme)

- Soit R_ϵ, l'ensemble des non-terminaux qui peuvent produire le mot vide.
- Soit $R_{i,l}$, l'ensemble des non-terminaux qui peuvent produire la sous-chaîne de s de longueur l à la position i, que nous noterons $s_{i,l}$.
  (Si $s \equiv t_0 \dots t_n$ alors $s_{i,l} = t_i \dots t_{i+l-1}$.)
- L'algorithme CYK est défini ainsi :

Pour l **de** 0 **à** n **faire**
  Pour chaque $A \to c_1 \dots c_m$ **de** G **faire**
   Pour i **de** 0 **à** $n - l$ **faire**
      Si on peut partager $s_{i,l}$ en m segments $(w_i)_{i \in 1 \dots m}$
      Tels que $\forall i, c_i$ produit w_i
      Alors
        $R_{i,l} \leftarrow R_{i,l} \cup \{ A \}$
   Fin
  Fin
Fin

Explication (hors programme)

> ...
> **Si** on peut partager $s_{i,l}$ en m segments $(w_i)_{i \in 1 \ldots m}$
> **Tels que** $\forall i, c_i$ produit w_i
> ...

- On doit tester tous les partitionnements de la sous-chaîne $s_{i,l}$ en m segments éventuellement vides.
- Pour tous les segments qui ont une taille k inférieure à n, on peut utiliser l'information des $R_{i,k}$.
- Premier problème : Si $m = 1$, et c_i est un non-terminal A_i alors on doit considérer les règles unitaires de la forme $A_i \rightarrow B$. Or, ce sont exactement celles que nous sommes en train de définir !

$\Rightarrow$ Encore un cas de **fermeture transitive** à calculer par itération.

- Second problème : Si tous les c_i, sauf un, produisent le mot vide, on se retrouve le même cas de figure que dans le premier problème.

Étude de complexité (hors programme)

- Soit n, la taille de l'entrée à analyser.
- Pour une règle dont le membre droit contient m symboles.
- On doit trouver $m-1$ points de partitionnement de l'entrée.
- Une fois trouvé, un point de partitionnement doit être combiné avec tout ceux qui le précèdent.
- Trouver un point de partitionnement peut demander $O(n)$ actions.
- Trouver tous les points de partitionnement nécessite donc $O(n^{m-1})$ actions.
- Il y a $O(n^2)$ cases dans la table, l'algorithme demande donc $O(n^{m+1})$ actions.

$\Rightarrow$ Pour la grammaire précédente, l'analyse est donc de complexité $O(n^4)$.

CYK sur les grammaires en forme normale de Chomsky (hors programme)

- Une grammaire est en forme normale de Chomsky si toutes ses règles sont de la forme :
  - « $A \to a$ » où a est un terminal.
  - « $A \to B\ C$ » où B et C sont des non-terminaux.

$\Rightarrow$ La complexité de CYK sur ces grammaires est “seulement” cubique.

$\Rightarrow$ L'absence de règles unitaires et de production vide rend inutiles les calculs de point fixe.

Mise en forme normale de Chomsky (hors programme)

- Nous savons déjà supprimer les règles unitaires et de production vide.
- Il reste alors des règles de la forme « $A \to c_1 \dots c_n$ » avec $n \geq 2$ et de la forme « $A \to a$ ».
- Pour tout c_i qui est un terminal a_i, on introduit un non-terminal T_i ainsi qu'une règle « $T_i \to a_i$ ». On peut remplacer chaque occurrence de c_i dans les règles de la première forme par le non-terminal T_i correspondant. Le langage reconnu par la grammaire est bien sûr préservé.
- Il reste alors des règles de la forme :

$$A \to A_1 \dots A_n$$

  pour $n \geq 3$
- Il suffit de les "découper" en multiples règles de la forme :

$$\begin{array}{rcl} A & \to & A_{1\_n-1} \dots A_n \\ A_{1\_n-1} & \to & A_{1\_n-2} \dots A_{n-1} \\ & \dots & \end{array}$$

Exemple de mise en forme normale de Chomsky (hors programme)

$$
\begin{array}{lcl}
S & ::= & / \; L \\
L & ::= & \epsilon \mid E \; / \; L \\
E & ::= & A \mid B \\
A & ::= & aa \mid a \\
B & ::= & bb \mid b
\end{array}
$$

Exercice

Mettre cette grammaire en forme normale de Chomsky.

Algorithme d'analyse ascendante dirigée

Une analyse directionnelle et ascendante

- On choisit, encore une fois, de se contraindre à une lecture de gauche à droite du mot à analyser.
- Sous cette contrainte, une analyse ascendante reconnaît nécessairement la dérivation droite dans l'ordre inverse de sa construction.

Une analyse directionnelle et ascendante

- Voici un exemple :

$$\begin{array}{lcl} S & ::= & A\ B\ C \\ A & ::= & a \\ B & ::= & b \\ C & ::= & c \end{array}$$

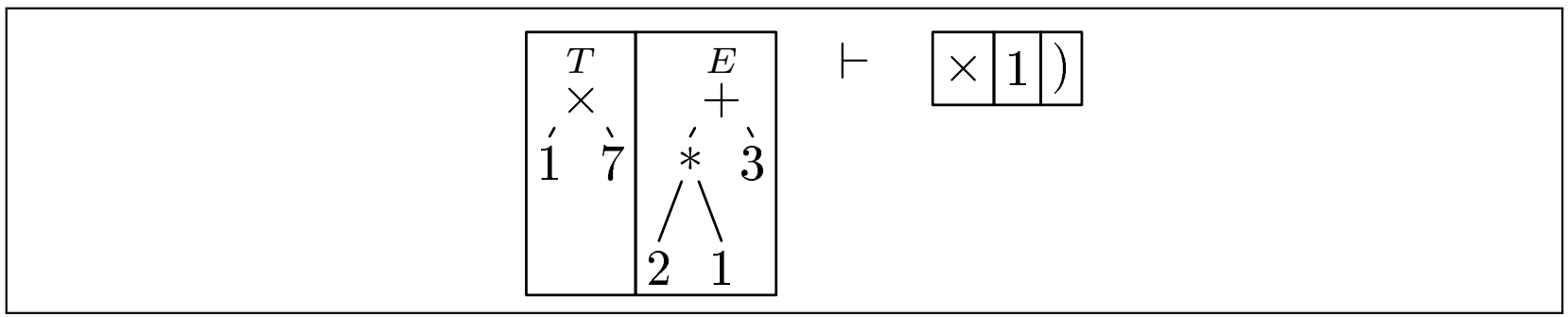
- L'analyse ascendante du mot « a b c » suit les étapes suivantes (• représente le pointeur de lecture) :

1. $\bullet\ a\ b\ c$
2. $A \bullet\ b\ c$
3. $A\ B \bullet\ c$
4. $A\ B\ C \bullet$
5. $S \bullet$

- En lisant cette séquence en sens inverse, on reconstruit bien la dérivation droite, c'est-à-dire celle qui réécrit le non-terminal le plus à droite.

Automate à pile pour l'analyse ascendante

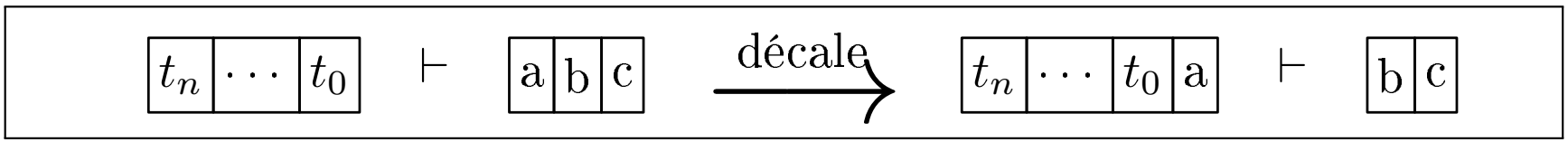
- Comme pour l'analyse descendante, l'automate à pile est un bon modèle de calcul pour l'analyse syntaxique ascendante.
- On représente une configuration d'analyse ascendante ainsi :



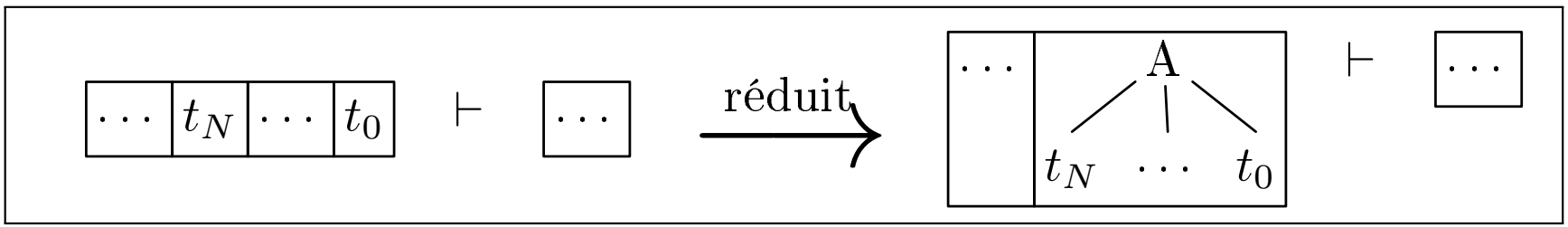
- À gauche du signe '$\vdash$' se trouve une pile des non-terminaux reconnus pour le moment avec leur arbre de production respectif. Le sommet de cette pile se trouve du côté du signe '$\vdash$'.
- À droite du signe '$\vdash$' se trouve le reste de l'entrée à analyser.

Actions de l'automate à pile pour l'analyse ascendante

- Il y a **deux** actions possibles : le décalage et la réduction.
- Décaler, c'est lire la première lettre de l'entrée et la poser sur la pile :



- Réduire, c'est dépiler N sous-arbres et empiler un nouvel arbre formé par l'application d'une règle de la grammaire :



- Notez que ces deux opérations sont **inversibles**.

Espace de recherche

- À tout moment, tant que l'entrée n'est pas consommée, l'automate peut effectuer une action « décale ».
- À tout moment, un certain nombre, positif ou nul, d'actions « réduire » sont aussi possibles.
- Une configuration est un échec, si aucune action n'est possible et que la pile n'est pas formée uniquement d'un arbre étiqueté par le non-terminal d'entrée de la grammaire.
- Une configuration est un succès, si la totalité de l'entrée est consommée et que la pile est formée d'un unique arbre étiqueté par le non-terminal d'entrée de la grammaire.

$\Rightarrow$ Ces informations définissent un espace de recherche.

Utilisation d'un foncteur générique

- Il existe une bibliothèque d'algorithmes de recherche écrite par J.C. Filliâtre : `http://www.lri.fr/~filliatr/ftp/ocaml/misc/search.ml.html`
- Notre domaine de recherche peut être écrit sous la forme d'un module OCaml dont la signature est :

```
module type ImperativeProblem = sig
  type move
  val success : unit → bool
  val moves : unit → move list
  val do_move : move → unit
  val undo_move : move → unit
  val add : unit → unit
  val mem : unit → bool
  val clear : unit → unit
end
```

Grammaire d'exemple

- Nous allons analyser le mot « aaaab » par la grammaire suivante :

$$
\begin{array}{lcl}
S & ::= & a\ N\ b \\
N & ::= & N\ a\ b \\
N & ::= & a\ a\ a
\end{array}
$$

Algorithme de recherche en profondeur

- Voici la trace obtenue pour une recherche en profondeur :

by shift	$\vdash$ a a a a b	a $\vdash$ a a a b
by shift	a $\vdash$ a a a b	a a $\vdash$ a a b
by shift	a a $\vdash$ a a b	a a a $\vdash$ a b
by shift	a a a $\vdash$ a b	a a a a $\vdash$ b
by shift	a a a a $\vdash$ b	a a a a b $\vdash$
rolling back shift	a a a a b $\vdash$	a a a a $\vdash$ b
by reducing top 3 elements using a a a to N		
	a a a a $\vdash$ b	a (N a a a) $\vdash$ b
by shift	a (N a a a) $\vdash$ b	a (N a a a) b $\vdash$
by reducing top 3 elements using a N b to S		
	a (N a a a) b $\vdash$	(S a (N a a a) b) $\vdash$

Algorithme de recherche en largeur

- Voici la trace obtenue pour une recherche en largeur :

by shift	⊢ a a a a b	a ⊢ a a a b
by shift	a ⊢ a a a b	a a ⊢ a a b
by shift	a a ⊢ a a b	a a a ⊢ a b
by shift	a a a ⊢ a b	a a a a ⊢ b
rolling back shift,	a a a a ⊢ b	a a a ⊢ a b
by reducing top 3 elements using a a a to N		
	a a a ⊢ a b	(N a a a) ⊢ a b
rolling back reducing top 3 elements using a a a to N		
	(N a a a) ⊢ a b	a a a ⊢ a b
by shift	a a a ⊢ a b	a a a a ⊢ b
by shift	a a a a ⊢ b	a a a a b ⊢
rolling back shift,	a a a a b ⊢	a a a a ⊢ b
by reducing top 3 elements using a a a to N		
	a a a a ⊢ b	a (N a a a) ⊢ b
rolling back reducing top 3 elements using a a a to N		
	a (N a a a) ⊢ b	a a a a ⊢ b
rolling back shift,	a a a a ⊢ b	a a a ⊢ a b
by reducing top 3 elements using a a a to N		
	a a a ⊢ a b	(N a a a) ⊢ a b
by shift	(N a a a) ⊢ a b	(N a a a) a ⊢ b
rolling back shift,	(N a a a) a ⊢ b	(N a a a) ⊢ a b
rolling back reducing top 3 elements using a a a to N		
	(N a a a) ⊢ a b	a a a ⊢ a b
by shift	a a a ⊢ a b	a a a a ⊢ b
by reducing top 3 elements using a a a to N		
	a a a a ⊢ b	a (N a a a) ⊢ b
by shift	a (N a a a) ⊢ b	a (N a a a) b ⊢
rolling back shift,	a (N a a a) b ⊢	a (N a a a) ⊢ b

rolling back reducing top 3 elements using a a a to N		
	a (N a a a) ⊢ b	a a a a ⊢ b
rolling back shift,	a a a a ⊢ b	a a a ⊢ a b
by reducing top 3 elements using a a a to N		
	a a a ⊢ a b	(N a a a) ⊢ a b
by shift	(N a a a) ⊢ a b	(N a a a) a ⊢ b
by shift	(N a a a) a ⊢ b	(N a a a) a b ⊢
rolling back shift,	(N a a a) a b ⊢	(N a a a) a ⊢ b
rolling back shift,	(N a a a) a ⊢ b	(N a a a) ⊢ a b
rolling back reducing top 3 elements using a a a to N		
	(N a a a) ⊢ a b	a a a ⊢ a b
by shift	a a a ⊢ a b	a a a a ⊢ b
by reducing top 3 elements using a a a to N		
	a a a a ⊢ b	a (N a a a) ⊢ b
by shift	a (N a a a) ⊢ b	a (N a a a) b ⊢
by reducing top 3 elements using a N b to S		
	a (N a a a) b ⊢	(S a (N a a a) b) ⊢

Grammaire d'exemple plus réaliste

- Combien d'étapes sont nécessaires à l'analyse du mot « a - a + a » par la grammaire suivante ?

$$
\begin{array}{lcl}
S & ::= & E \\
E & ::= & E\ Q\ F \mid F \\
F & ::= & a \\
Q & ::= & + \mid -
\end{array}
$$

$\Rightarrow$ 292 pour la recherche en profondeur, 1369 pour la recherche en largeur !

Observation

- Si on regarde de plus près les chemins suivis par l'algorithme dans le domaine de recherche, on en trouve de la forme :

by shift	$\vdash$ a - a + a	a $\vdash$ - a + a
by shift	a $\vdash$ - a + a	a - $\vdash$ a + a
by shift	a - $\vdash$ a + a	a - a $\vdash$ + a
by shift	a - a $\vdash$ + a	a - a + $\vdash$ a
by shift	a - a + $\vdash$ a	a - a + a $\vdash$
by reducing top 1 elements using a to F		
	a - a + a $\vdash$	a - a + (F a) $\vdash$
by reducing top 1 elements using F to E		
	a - a + (F a) $\vdash$	a - a + (E (F a)) $\vdash$
by reducing top 1 elements using E to S		
	a - a + (E (F a)) $\vdash$	a - a + (S (E (F a))) $\vdash$

- Cette dernière réduction est absurde puisque S ne peut produire un E qu'à la fin du mot !

Algorithme de Earley

Présentation

- L'algorithme de Earley coupe des branches de l'arbre de recherche en prenant en compte le **contexte** dans lequel l'analyse ascendante est effectuée.
- Dans l'exemple précédent :

by shift	$\vdash$ a - a + a	a $\vdash$ - a + a
by shift	a $\vdash$ - a + a	a - $\vdash$ a + a
by shift	a - $\vdash$ a + a	a - a $\vdash$ + a
by shift	a - a $\vdash$ + a	a - a + $\vdash$ a
by shift	a - a + $\vdash$ a	a - a + a $\vdash$
by reducing top 1 elements using a to F		
	a - a + a $\vdash$	a - a + (F a) $\vdash$
by reducing top 1 elements using F to E		
	a - a + (F a) $\vdash$	a - a + (E (F a)) $\vdash$
by reducing top 1 elements using E to S		
	a - a + (E (F a)) $\vdash$	a - a + (S (E (F a))) $\vdash$

- Après avoir vu passer le terminal '$-$', on peut initier la reconnaissance d'un non-terminal F mais on sait qu'un E et un S ne peuvent pas être reconnus.

$\Rightarrow$ L'analyse ascendante doit être dirigée par une analyse descendante.

Les “états d’analyse” de l’algorithme de Earley

- L’algorithme de Earley utilise une description de l’état de l’analyse, appelé *item* en anglais, qui suit la syntaxe suivante :

$$N \to s_1 \dots s_k \bullet s'_1 \dots s'_n$$

où les s_i et les s'_j sont des symboles terminaux ou non-terminaux.

- Cette description se lit :

> *« Au sein de l’analyse du non-terminal N, les symboles $s_1, \dots, s_k$ ont été reconnus et les symboles $s'_1, \dots, s'_n$ sont attendus par la suite (ils sont prédits par l’analyse).»*

Classification des états de l'analyse

- La position du symbole $\bullet$ caractérise différents stades de l'analyse.
- État à réduire « $N \rightarrow s_1 \dots s_k \bullet$ » :
  L'analyse de N est terminée. On peut légitimement réduire les k éléments du sommet de la pile en utilisant cette règle. L'analyse peut se poursuivre dans l'état qui a initié l'analyse du non-terminal N.
- État prédit « $N \rightarrow \bullet\, s'_1 \dots s'_n$ » :
  L'analyse de N débute. Cet état a été initié par une prédiction.
- État à décaler « $N \rightarrow s_1 \dots s_k \bullet a \dots s'_n$ » :
  L'analyse prédit un terminal. L'automate peut seulement effectuer l'action « décale ».
- État de prédiction « $N \rightarrow s_1 \dots s_k \bullet N' \dots s'_n$ » :
  L'analyse prédit un non-terminal. On doit considérer l'ensemble des états prédits qui ont un sens en ce point.

Description étendue d'état d'analyse

- Pour être capable d'associer un état prédit à son état de prédiction, on utilise la **position dans le mot à analyser** correspondant à la position du non-terminal reconnu par l'état prédit.
- En d'autres termes, on associe un entier à chaque description d'état d'analyse qui correspond à la position à partir de laquelle cette analyse a été initiée.
- On note ainsi la description étendue que l'on obtient :

$$N \rightarrow s_1 \dots s_k \bullet s'_1 \dots s'_n \quad @\, p$$

- Elle se lit :

  « Au sein de l'analyse du non-terminal N ***initiée à la position*** *p* ***de l'entrée****, les symboles $s_1, \dots, s_k$ ont été reconnus et les symboles $s'_1, \dots, s'_n$ sont attendus par la suite (on dit aussi qu'ils sont prédits par l'analyse).»*

L'algorithme de Earley (version naïve) (hors programme)

- L'algorithme de Earley est décomposable en trois fonctions.
- Ces fonctions travaillent sur quatre ensembles de descriptions d'état d'analyse :
  - Complétés_p : l'ensemble des états à réduire à la position p.
  - Actifs_p : l'ensemble des états à décaler ou en prédiction à la position p.
  - Prédits_p : l'ensemble des états prédits à la position p.
  - Tous_p : tous les états de la position p.

L'algorithme de Earley (version naïve) (hors programme)

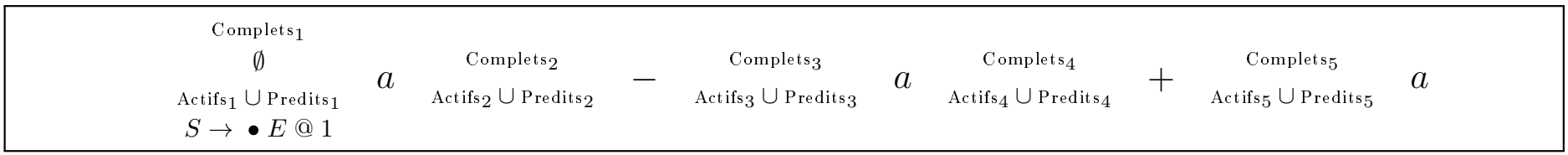
- L'algorithme de Earley est décomposable en trois fonctions.
- Fonction de lecture de l'entrée « `scan` » :
  Transforme les états actifs à décaler à la position p en états de la position $p+1$. Ces états peuvent être à réduire, à décaler ou de prédiction.
- Fonction de réduction « `complete` » :
  Traite les états complétés à la position p dont l'analyse a été initiée en m. Appelle la fonction `scan` sur tous les états actifs en position m qui ont initié l'analyse du non-terminal reconnu en p. Cette fonction décale ces états en p.
  (Plusieurs itérations peuvent être nécessaires.)
- Fonction de prédiction « `predict` » :
  Pour chaque état de prédiction actif en p dont le non-terminal prédit en tête est N, rajoute dans $\mathrm{Prédits}_p$ les états « $N \to \bullet s_1 \dots s_n @ p$ » pour chaque règle de la forme « $N \to s_1 \dots s_n$ » dans la grammaire. (Plusieurs itérations peuvent être nécessaires.)

Exemple

- Analysons le mot “a - a + a” à travers la grammaire :

$$
\begin{array}{lcl}
S & ::= & E \\
E & ::= & E\ Q\ F \mid F \\
F & ::= & a \\
Q & ::= & + \mid -
\end{array}
$$

Exemple (hors programme)



- Les règles du symbole d'entrée produisent les états d'analyse actifs initiaux.

Exemple (hors programme)

$$\begin{array}{c} \text{Complets}_1 \\ \emptyset \\ \text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E \;@\, 1 \\ E \to \bullet\, E\; Q\; F \;@\, 1 \\ F \to \bullet\, F \;@\, 1 \\ F \to \bullet\, a \;@\, 1 \end{array} \quad a \quad \begin{array}{c} \text{Complets}_2 \\ \text{Actifs}_2 \cup \text{Predits}_2 \end{array} \quad - \quad \begin{array}{c} \text{Complets}_3 \\ \text{Actifs}_3 \cup \text{Predits}_3 \end{array} \quad a \quad \begin{array}{c} \text{Complets}_4 \\ \text{Actifs}_4 \cup \text{Predits}_4 \end{array} \quad + \quad \begin{array}{c} \text{Complets}_5 \\ \text{Actifs}_5 \cup \text{Predits}_5 \end{array} \quad a$$

- La fonction `predict` rajoute toutes les expansions possibles de E.

Exemple (hors programme)

$$\underset{\emptyset}{\overset{\text{Complets}_1}{}} \quad \underset{\begin{array}{l} S \to \bullet\, E\ @\,1 \\ E \to \bullet\, E\ Q\ F\ @\,1 \\ E \to \bullet\, F\ @\,1 \\ F \to \bullet\, a\ @\,1 \end{array}}{\overset{\text{Actifs}_1 \cup \text{Predits}_1}{}} \quad a \quad \underset{F \to a\ \bullet\ @\,1}{\overset{\text{Complets}_2}{}} \quad \underset{\text{Actifs}_2 \cup \text{Predits}_2}{} \quad - \quad \underset{\text{Actifs}_3 \cup \text{Predits}_3}{\overset{\text{Complets}_3}{}} \quad a \quad \underset{\text{Actifs}_4 \cup \text{Predits}_4}{\overset{\text{Complets}_4}{}} \quad + \quad \underset{\text{Actifs}_5 \cup \text{Predits}_5}{\overset{\text{Complets}_5}{}} \quad a$$

- La fonction `scan` décale le terminal en tête *a* et insère l'état d'analyse obtenu dans l'ensemble Complets_2.

Exemple (hors programme)

$$\underset{\emptyset}{\underset{\text{Actifs}_1 \cup \text{Predits}_1}{\underset{S \to \bullet E\ @\ 1}{\underset{E \to \bullet E\ Q\ F\ @\ 1}{\underset{E \to \bullet F\ @\ 1}{\underset{F \to \bullet a\ @\ 1}{\text{Complets}_1}}}}}}\ a\ \underset{\text{Actifs}_2 \cup \text{Predits}_2}{\underset{F \to a\ \bullet\ @\ 1}{\underset{E \to F\ \bullet\ @\ 1}{\text{Complets}_2}}}\ -\ \underset{\text{Actifs}_3 \cup \text{Predits}_3}{\text{Complets}_3}\ a\ \underset{\text{Actifs}_4 \cup \text{Predits}_4}{\text{Complets}_4}\ +\ \underset{\text{Actifs}_5 \cup \text{Predits}_5}{\text{Complets}_5}\ a$$

- La fonction `complete` réduit le terminal *a* sur la pile en un non-terminal *F* puis appelle la fonction `scan` pour qu'elle tienne compte de cette réduction.

Exemple (hors programme)

$$\underset{\emptyset}{\overset{\text{Complets}_1}{}}\;\underset{\begin{array}{l}S \to \bullet\, E\ @\,1\\ E \to \bullet\, E\ Q\ F\ @\,1\\ E \to \bullet\, F\ @\,1\\ F \to \bullet\, a\ @\,1\end{array}}{\overset{\text{Actifs}_1 \cup \text{Predits}_1}{}}\;a\;\underset{\begin{array}{l}F \to a\ \bullet\ @\,1\\ E \to F\ \bullet\ @\,1\\ S \to E\ \bullet\ @\,1\end{array}}{\overset{\text{Complets}_2}{}}\;\underset{E \to E\ \bullet\ Q\ F\ @\,1}{\overset{\text{Actifs}_2 \cup \text{Predits}_2}{}}\;-\;\underset{\text{Actifs}_3 \cup \text{Predits}_3}{\overset{\text{Complets}_3}{}}\;a\;\underset{\text{Actifs}_4 \cup \text{Predits}_4}{\overset{\text{Complets}_4}{}}\;+\;\underset{\text{Actifs}_5 \cup \text{Predits}_5}{\overset{\text{Complets}_5}{}}\;a$$

- Par itération de cette interaction entre la fonction `scan` et `complete`, on complète l'ensemble Tous_2.

Exemple (hors programme)

$$\underset{\substack{\text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E\ @\,1 \\ E \to \bullet\, E\, Q\, F\ @\,1 \\ E \to \bullet\, F\ @\,1 \\ F \to \bullet\, a\ @\,1}}{\underset{\text{Complets}_1}{\emptyset}} \; a \; \underset{\substack{F \to a\ \bullet\ @\,1 \\ E \to F\ \bullet\ @\,1 \\ S \to E\ \bullet\ @\,1}}{\underset{\substack{\text{Actifs}_2 \cup \text{Predits}_2 \\ E \to E\ \bullet\ Q\, F\ @\,1 \\ Q \to \bullet\, +\ @\,2 \\ Q \to \bullet\, -\ @\,2}}{\underset{\text{Complets}_2}{}}} \; - \; \underset{\text{Actifs}_3 \cup \text{Predits}_3}{\underset{\text{Complets}_3}{}} \; a \; \underset{\text{Actifs}_4 \cup \text{Predits}_4}{\underset{\text{Complets}_4}{}} \; + \; \underset{\text{Actifs}_5 \cup \text{Predits}_5}{\underset{\text{Complets}_5}{}} \; a$$

- Il apparaît alors de nouveaux états de prédiction à traiter par la fonction `predict`.

Exemple (hors programme)

$$\underset{\substack{\text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E \;@\, 1 \\ E \to \bullet\, E\, Q\, F \;@\, 1 \\ E \to \bullet\, F \;@\, 1 \\ F \to \bullet\, a \;@\, 1}}{\text{Complets}_1 \\ \emptyset} \; a \; \underset{\substack{\text{Actifs}_2 \cup \text{Predits}_2 \\ E \to E\, \bullet\, Q\, F \;@\, 1 \\ Q \to \bullet\, + \;@\, 2 \\ Q \to \bullet\, - \;@\, 2}}{\text{Complets}_2 \\ \begin{array}{l} F \to a\, \bullet \;@\, 1 \\ E \to F\, \bullet \;@\, 1 \\ S \to E\, \bullet \;@\, 1 \end{array}} \; - \; \underset{\substack{\text{Actifs}_3 \cup \text{Predits}_3 \\ E \to E\, Q\, \bullet\, F \;@\, 1 \\ F \to \bullet\, a \;@\, 3}}{\text{Complets}_3 \\ Q \to -\, \bullet \;@\, 2} \; a \; \underset{\text{Actifs}_4 \cup \text{Predits}_4}{\text{Complets}_4} \; + \; \underset{\text{Actifs}_5 \cup \text{Predits}_5}{\text{Complets}_5} \; a$$

- On suit ce procédé pour traiter l'ensemble du mot à analyser.
- À la fin de la lecture du mot d'entrée, on doit trouver un état à réduire vers le symbole d'entrée de la grammaire.

Exemple (hors programme)

$$\underset{\substack{\text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E\ @\,1 \\ E \to \bullet\, E\, Q\, F\ @\,1 \\ E \to \bullet\, F\ @\,1 \\ F \to \bullet\, a\ @\,1}}{\text{Complets}_1} \quad a \quad \underset{\substack{F \to a\ \bullet\ @\,1 \\ E \to F\ \bullet\ @\,1 \\ S \to E\ \bullet\ @\,1 \\ \text{Actifs}_2 \cup \text{Predits}_2 \\ E \to E\ \bullet\ Q\, F\ @\,1 \\ Q \to \bullet\, +\ @\,2 \\ Q \to \bullet\, -\ @\,2}}{\text{Complets}_2} \quad - \quad \underset{\substack{Q \to -\ \bullet\ @\,2 \\ \text{Actifs}_3 \cup \text{Predits}_3 \\ E \to E\, Q\ \bullet\ F\ @\,1 \\ F \to \bullet\, a\ @\,3}}{\text{Complets}_3} \quad a \quad \underset{\substack{F \to a\ \bullet\ @\,3 \\ E \to E\, Q\, F\ \bullet\ @\,1 \\ S \to E\ \bullet\ @\,1 \\ \text{Actifs}_4 \cup \text{Predits}_4 \\ E \to E\ \bullet\ Q\, F\ @\,1 \\ Q \to \bullet\, +\ @\,4 \\ Q \to \bullet\, -\ @\,4}}{\text{Complets}_4} \quad + \quad \underset{\text{Actifs}_5 \cup \text{Predits}_5}{\text{Complets}_5} \quad a$$

- On suit ce procédé pour traiter l'ensemble du mot à analyser.
- À la fin de la lecture du mot d'entrée, on doit trouver un état à réduire vers le symbole d'entrée de la grammaire.

Exemple (hors programme)

$$\underset{\substack{\text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E \;@\,1 \\ E \to \bullet\, E\, Q\, F \;@\,1 \\ E \to \bullet\, F \;@\,1 \\ F \to \bullet\, a \;@\,1}}{\text{Complets}_1 \\ \emptyset} \quad a \quad \underset{\substack{F \to a \;\bullet\; @\,1 \\ E \to F \;\bullet\; @\,1 \\ S \to E \;\bullet\; @\,1 \\ \text{Actifs}_2 \cup \text{Predits}_2 \\ E \to E \;\bullet\; Q\, F \;@\,1 \\ Q \to \bullet\, + \;@\,2 \\ Q \to \bullet\, - \;@\,2}}{\text{Complets}_2} \quad - \quad \underset{\substack{Q \to - \;\bullet\; @\,2 \\ \text{Actifs}_3 \cup \text{Predits}_3 \\ E \to E\, Q \;\bullet\; F \;@\,1 \\ F \to \bullet\, a \;@\,3}}{\text{Complets}_3} \quad a \quad \underset{\substack{F \to a \;\bullet\; @\,3 \\ E \to E\, Q\, F \;\bullet\; @\,1 \\ S \to E \;\bullet\; @\,1 \\ \text{Actifs}_4 \cup \text{Predits}_4 \\ E \to E \;\bullet\; Q\, F \;@\,1 \\ Q \to \bullet\, + \;@\,4 \\ Q \to \bullet\, - \;@\,4}}{\text{Complets}_4} \quad + \quad \underset{\substack{Q \to + \;\bullet\; @\,4 \\ \text{Actifs}_5 \cup \text{Predits}_5 \\ E \to E\, Q \;\bullet\; F \;@\,1 \\ F \to \bullet\, a \;@\,5}}{\text{Complets}_5} \quad a$$

- On suit ce procédé pour traiter l'ensemble du mot à analyser.
- À la fin de la lecture du mot d'entrée, on doit trouver un état à réduire vers le symbole d'entrée de la grammaire.

Exemple (hors programme)

Complets_1
$\emptyset$
$\text{Actifs}_1 \cup \text{Predits}_1$
$S \to \bullet\, E \;@\, 1$
$E \to \bullet\, E\, Q\, F \;@\, 1$
$E \to \bullet\, F \;@\, 1$
$F \to \bullet\, a \;@\, 1$

a

Complets_2
$F \to a \;\bullet\; @\, 1$
$E \to F \;\bullet\; @\, 1$
$S \to E \;\bullet\; @\, 1$
$\text{Actifs}_2 \cup \text{Predits}_2$
$E \to E \;\bullet\; Q\, F \;@\, 1$
$Q \to \bullet\, + \;@\, 2$
$Q \to \bullet\, - \;@\, 2$

$-$

Complets_3
$Q \to - \;\bullet\; @\, 2$
$\text{Actifs}_3 \cup \text{Predits}_3$
$E \to E\, Q \;\bullet\; F \;@\, 1$
$F \to \bullet\, a \;@\, 3$

a

Complets_4
$F \to a \;\bullet\; @\, 3$
$E \to E\, Q\, F \;\bullet\; @\, 1$
$S \to E \;\bullet\; @\, 1$
$\text{Actifs}_4 \cup \text{Predits}_4$
$E \to E \;\bullet\; Q\, F \;@\, 1$
$Q \to \bullet\, + \;@\, 4$
$Q \to \bullet\, - \;@\, 4$

$+$

Complets_5
$Q \to + \;\bullet\; @\, 4$
$\text{Actifs}_5 \cup \text{Predits}_5$
$E \to E\, Q \;\bullet\; F \;@\, 1$
$F \to \bullet\, a \;@\, 5$

a

Complets_6
$F \to a \;\bullet\; @\, 5$
$E \to E\, Q\, F \;\bullet\; @\, 1$
$S \to E \;\bullet\; @\, 1$
$\text{Actifs}_6 \cup \text{Predits}_6$
$E \to E \;\bullet\; Q\, F \;@\, 1$

- On suit ce procédé pour traiter l'ensemble du mot à analyser.
- À la fin de la lecture du mot d'entrée, on doit trouver un état à réduire vers le symbole d'entrée de la grammaire.

Problème des règles de production vide (hors programme)

- Encore une fois, les règles de production vide posent problème.
- Analysons le mot « aa / a » à travers la grammaire suivante :

$$
\begin{array}{lcl}
S & ::= & E \\
E & ::= & E\ Q\ F \mid F \\
F & ::= & a \\
Q & ::= & \times \mid / \mid \epsilon
\end{array}
$$

Exemple problématique (hors programme)

$$\begin{array}{c} \text{Complets}_1 \\ \emptyset \\ \text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E \;@\, 1 \end{array} \quad a \quad \begin{array}{c} \text{Complets}_2 \\ \text{Actifs}_2 \cup \text{Predits}_2 \end{array} \quad a \quad \begin{array}{c} \text{Complets}_4 \\ \text{Actifs}_4 \cup \text{Predits}_4 \end{array} \quad / \quad \begin{array}{c} \text{Complets}_5 \\ \text{Actifs}_5 \cup \text{Predits}_5 \end{array} \quad a$$

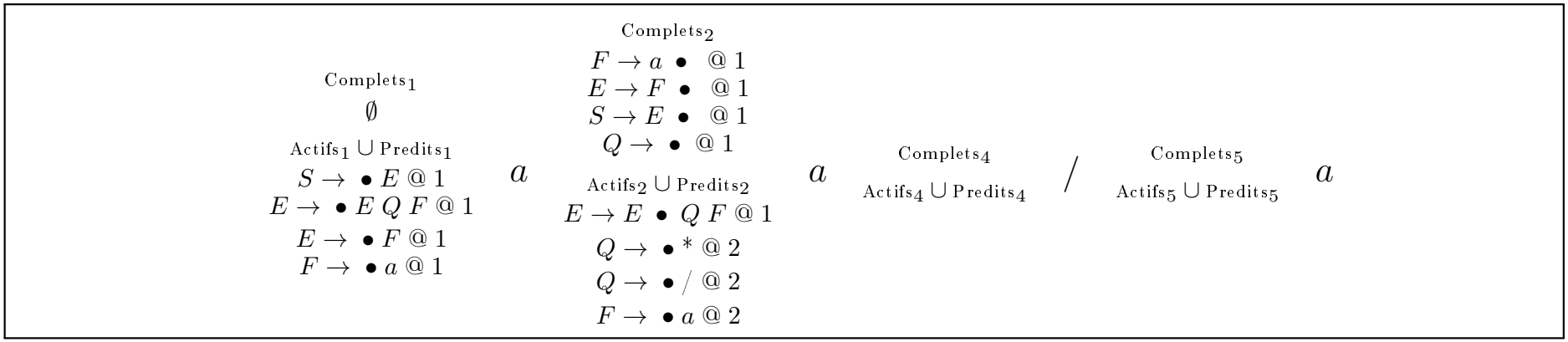
- Les premières étapes ne posent pas de problème.

Exemple problématique (hors programme)

$$\begin{array}{c} \text{Complets}_1 \\ \emptyset \\ \text{Actifs}_1 \cup \text{Predits}_1 \\ S \to \bullet\, E \;@\, 1 \\ E \to \bullet\, E\, Q\, F \;@\, 1 \\ E \to \bullet\, F \;@\, 1 \\ F \to \bullet\, a \;@\, 1 \end{array} \quad a \quad \begin{array}{c} \text{Complets}_2 \\ \text{Actifs}_2 \cup \text{Predits}_2 \end{array} \quad a \quad \begin{array}{c} \text{Complets}_4 \\ \text{Actifs}_4 \cup \text{Predits}_4 \end{array} \quad / \quad \begin{array}{c} \text{Complets}_5 \\ \text{Actifs}_5 \cup \text{Predits}_5 \end{array} \quad a$$

- Les premières étapes ne posent pas de problème.

Exemple problématique (hors programme)

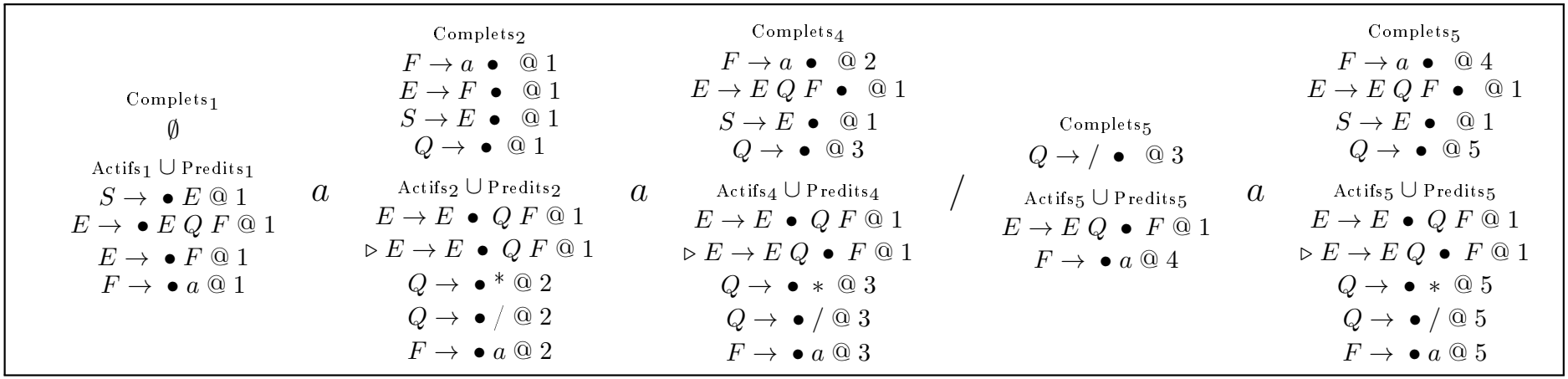


- La fonction `predict` doit rajouter l'état « $Q \to \epsilon \bullet$ » à l'ensemble $\mathrm{Complets}_2$.
- Or, cet ensemble est censé être déjà déterminé par la fonction `complete`.

$\Rightarrow$ On itère une alternance d'application des fonctions `complete` et `predict`.

$\Rightarrow$ Heureusement, ce processus termine. (Pourquoi ?)

Exemple problématique (hors programme)



- La fonction `predict` doit rajouter l'état « $Q \to \epsilon \bullet$ » à l'ensemble $\mathrm{Complets}_2$.
- Or, cet ensemble est censé être déjà déterminé par la fonction `complete`.

$\Rightarrow$ On itère une alternance d'application des fonctions `complete` et `predict`.

$\Rightarrow$ Heureusement, ce processus termine. (Pourquoi ?)

Suppression de l'itération `complete/predict` (hors programme)

- On peut se passer de l'itération précédente en réorganisant l'algorithme et en modifiant légèrement la fonction `predict`.
- On calcule, pour chaque position p, une liste ℓ_p définie à l'aide de ℓ_{p-1}.
- Initialement, ℓ_p contient les états d'analyse insérés par la fonction `scan` à l'itération précédente.
- Pour chaque état de la liste ℓ_p, si :
  - c'est un état à décaler, alors on applique la fonction `scan`.
  - c'est un état à réduire, alors on applique la fonction `complete`.
  - c'est un état de prédiction, alors on applique la fonction `predict`.

  $\Rightarrow$ Si ces deux dernières fonctions doivent insérées un nouvel état à la position p, cet état est rajouté **à la fin de** ℓ_p.
- La fonction `predict`, confrontée à un état d'analyse de la forme « $N \to \ldots \bullet B \ldots$ » où B peut produire le mot vide, produit, en plus des prédictions d'expansion de B, un état « $N \to \ldots B \bullet \ldots$ » inséré à la fin de ℓ_p.

Équivalence entre les deux algorithmes (hors programme)

- La preuve d'équivalence entre les deux algorithmes, l'un utilisant une itération et l'autre non, n'est pas immédiate.
- En fait, dans le second algorithme, l'itération est déplacée dans la fonction de calcul des non-terminaux produisant le mot vide.

Étude de complexité (hors programme)

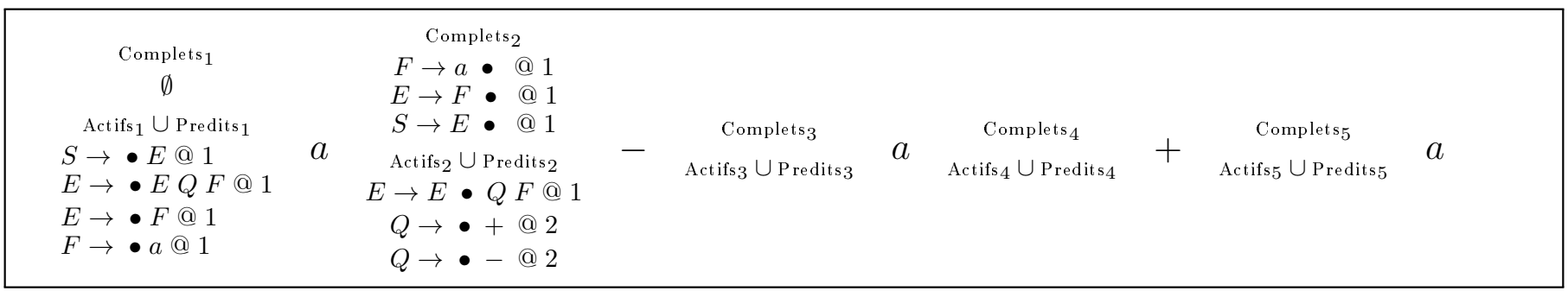
Exercice

Montrer que l'algorithme de Earley est cubique, en pire cas.

Dirigée la prédiction par le prochain terminal de l'entrée

(hors programme)

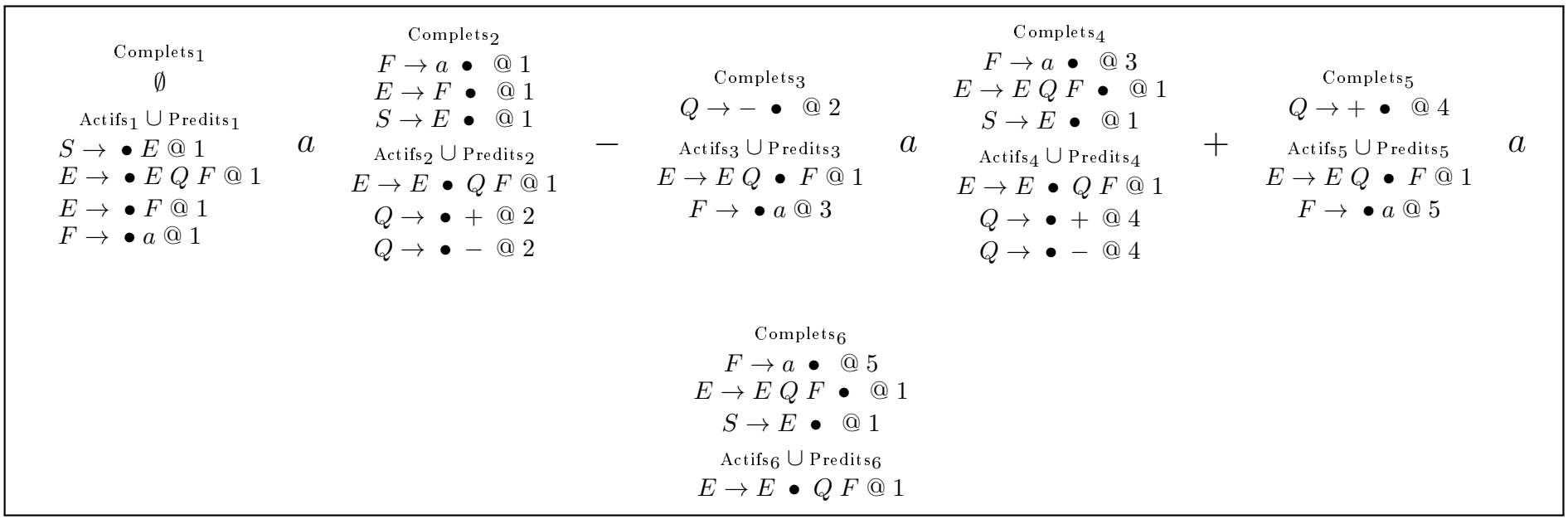
- Comme dans le cas des analyses ascendantes, on remarque qu'une grande partie des prédictions peut être éliminée par une observation du terminal suivant de l'entrée.
- À l'étape suivante de notre exemple :



- La prédiction « $Q \rightarrow \bullet + @2$ » est idiote car le prochain terminal est "-".

$\Rightarrow$ On peut généraliser cette idée en utilisant l'ensemble *FIRST*.

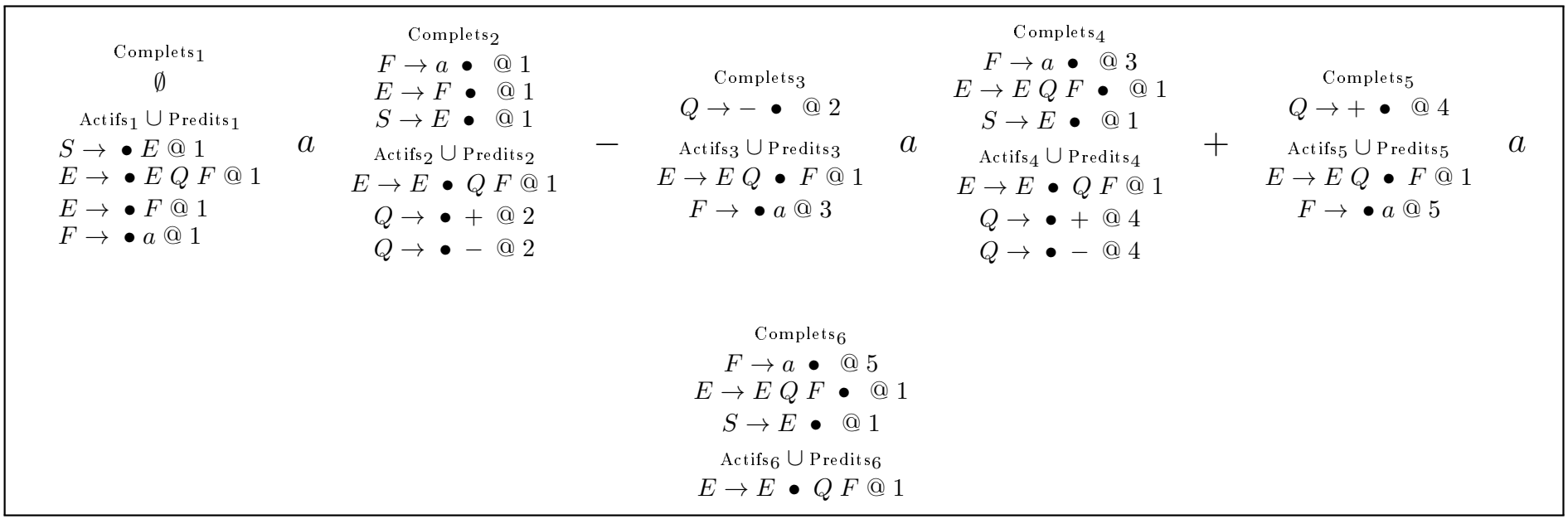
Application (hors programme)



Exercice

Quelles sont les prédictions idiotes ?

Encore une amélioration ? (hors programme)



Exercice

Quels sont les états à réduire éliminables par observation du prochain terminal ?

Dirigée la réduction par le prochain terminal de l'entrée

(hors programme)

- Pour l'analyse LL(1), nous avons calculé l'ensemble $FOLLOW(N)$ des terminaux qui peuvent suivrent le non-terminal N dans l'entrée.
- Nous pouvons réutiliser cette information pour étendre de nouveau la description des états d'analyse en lui rajoutant un ensemble de terminaux suivants potentiels :

$$N \to s_1 \dots s_k \bullet s'_1 \dots s'_n \quad [a_1 \dots a_m]@\, p$$

avec $FOLLOW(N) = \{a_1 \dots a_m\}$

- Il ne sert à rien de réduire un état de $\mathrm{Complets}_p$ si le prochain terminal n'est pas dans son ensemble de terminaux suivants potentiels.

Trace finale de l'algorithme de Earley (hors programme)

Complets_1

$\emptyset$

$\text{Actifs}_1 \cup \text{Predits}_1$

$S \to \bullet\, E\ [\#]\ @\ 1$

$E \to \bullet\, E\ Q\ F\ [\#\ +\ -]\ @\ 1$

$E \to \bullet\, F\ [\#\ +\ -]\ @\ 1$

$F \to \bullet\, a\ [\#\ +\ -]\ @\ 1$

a

Complets_2

$F \to a\ \bullet\ [\#\ +\ -]\ @\ 1$

$E \to F\ \bullet\ [\#\ +\ -]\ @\ 1$

$S \to E\ \bullet\ [\#]\ @\ 1$

$\text{Actifs}_2 \cup \text{Predits}_2$

$E \to E\ \bullet\ Q\ F\ [\#\ +\ -]\ @\ 1$

$Q \to \bullet\, +\ [a]\ @\ 2$

$Q \to \bullet\, -\ [a]\ @\ 2$

$-$

Complets_3

$Q \to -\ \bullet\ [a]\ @\ 2$

$\text{Actifs}_3 \cup \text{Predits}_3$

$E \to E\ Q\ \bullet\ F\ [\#\ +\ -]\ @\ 1$

$F \to \bullet\, a\ [\#\ +\ -]\ @\ 3$

a

Complets_4

$F \to a\ \bullet\ [\#\ +\ -]\ @\ 3$

$E \to E\ Q\ F\ \bullet\ [\#\ +\ -]\ @\ 1$

$S \to E\ \bullet\ [\#]\ @\ 1$

$\text{Actifs}_4 \cup \text{Predits}_4$

$E \to E\ \bullet\ Q\ F\ [\#\ +\ -]\ @\ 1$

$Q \to \bullet\, +\ [a]\ @\ 4$

$Q \to \bullet\, -\ [a]\ @\ 4$

$+$

Complets_5

$Q \to +\ \bullet\ [a]\ @\ 4$

$\text{Actifs}_5 \cup \text{Predits}_5$

$E \to E\ Q\ \bullet\ F\ [\#\ +\ -]\ @\ 1$

$F \to \bullet\, a\ [\#\ +\ -]\ @\ 5$

a

Complets_6

$F \to a\ \bullet\ [\#\ +\ -]\ @\ 5$

$E \to E\ Q\ F\ \bullet\ [\#\ +\ -]\ @\ 1$

$S \to E\ \bullet\ [\#]\ @\ 1$

$\text{Actifs}_6 \cup \text{Predits}_6$

$E \to E\ \bullet\ Q\ F\ [\#\ +\ -]\ @\ 1$

$Q \to \bullet\, +\ [a]\ @\ 6$

$Q \to \bullet\, -\ [a]\ @\ 6$

$\#$

Trace finale de l'algorithme de Earley (hors programme)

$$
\begin{array}{c}
\text{Complets}_1 \\
\emptyset \\
\text{Actifs}_1 \cup \text{Predits}_1 \\
S \to \bullet\, E\ [\#]\ @\ 1 \\
E \to \bullet\, E\ Q\ F\ [\#\ +\ -]\ @\ 1 \\
E \to \bullet\, F\ [\#\ +\ -]\ @\ 1 \\
F \to \bullet\, a\ [\#\ +\ -]\ @\ 1
\end{array}
\quad a \quad
\begin{array}{c}
\text{Complets}_2 \\
F \to a\ \bullet\ [\#\ +\ -]\ @\ 1 \\
E \to F\ \bullet\ [\#\ +\ -]\ @\ 1 \\
\text{Actifs}_2 \cup \text{Predits}_2 \\
E \to E\ \bullet\, Q\ F\ [\#\ +\ -]\ @\ 1 \\
Q \to \bullet\, +\ [a]\ @\ 2 \\
Q \to \bullet\, -\ [a]\ @\ 2
\end{array}
\quad - \quad
\begin{array}{c}
\text{Complets}_3 \\
Q \to -\ \bullet\ [a]\ @\ 2 \\
\text{Actifs}_3 \cup \text{Predits}_3 \\
E \to E\ Q\ \bullet\, F\ [\#\ +\ -]\ @\ 1 \\
F \to \bullet\, a\ [\#\ +\ -]\ @\ 3
\end{array}
\quad a
$$

$$
\begin{array}{c}
\text{Complets}_4 \\
F \to a\ \bullet\ [\#\ +\ -]\ @\ 3 \\
E \to E\ Q\ F\ \bullet\ [\#\ +\ -]\ @\ 1 \\
\text{Actifs}_4 \cup \text{Predits}_4 \\
E \to E\ \bullet\, Q\ F\ [\#\ +\ -]\ @\ 1 \\
Q \to \bullet\, +\ [a]\ @\ 4 \\
Q \to \bullet\, -\ [a]\ @\ 4
\end{array}
\quad + \quad
\begin{array}{c}
\text{Complets}_5 \\
Q \to +\ \bullet\ [a]\ @\ 4 \\
\text{Actifs}_5 \cup \text{Predits}_5 \\
E \to E\ Q\ \bullet\, F\ [\#\ +\ -]\ @\ 1 \\
F \to \bullet\, a\ [\#\ +\ -]\ @\ 5
\end{array}
\quad a \quad
\begin{array}{c}
\text{Complets}_6 \\
F \to a\ \bullet\ [\#\ +\ -]\ @\ 5 \\
E \to E\ Q\ F\ \bullet\ [\#\ +\ -]\ @\ 1 \\
S \to E\ \bullet\ [\#]\ @\ 1 \\
\text{Actifs}_6 \cup \text{Predits}_6 \\
E \to E\ \bullet\, Q\ F\ [\#\ +\ -]\ @\ 1 \\
Q \to \bullet\, +\ [a]\ @\ 6 \\
Q \to \bullet\, -\ [a]\ @\ 6
\end{array}
\quad \#
$$

Synthèse

Synthèse

- L'algorithme CYK nécessite une grammaire en forme normale de Chomsky pour effectuer une analyse syntaxique en $O(n^3)$.
- Les deux actions d'un automate à pile d'analyse ascendante sont « décale » et « réduit ». Elles forment un espace de recherche que l'on peut parcourir en temps exponentiel.
- L'algorithme de Earley élague cet arbre de recherche en étant dirigé par une analyse descendante paralèlle. Il effectue ainsi une analyse syntaxique en $O(n^3)$.

$\Rightarrow$ La prochaine fois, pour le dernier cours sur l'analyse syntaxique, nous étudierons les algorithmes de la famille LR, qui sont linéaires et utilisés par les outils de type YACC tels que MENHIR.

QCM : Question 1

Dans l'algorithme CYK, la case (i, l) de la table contient :

- ☐ le sous-mot de longueur l à la position i.
- ☐ le non-terminal qui peut produire le sous-mot de longueur l à la position i.
- ☐ l'ensemble des non-terminaux qui peuvent produire le sous-mot de longueur l à la position i.
- ☐ la pile de l'automate utilisé pour analyser le sous-mot à la position i et de longueur l.

QCM : Question 2

La complexité en pire cas de l'algorithme CYK est

- ☐ $O(n^4)$
- ☐ $O(n^3)$
- ☐ $O(n^{m+1})$ avec m la longueur maximale des règles de la grammaire.
- ☐ $O(n)$

QCM : Question 3

À propos des grammaires de Chomsky, quelles affirmations sont vraies :

- ☐ On peut associer une forme normale de Chomsky à toute grammaire hors-contexte.
- ☐ L'algorithme CYK est quadratique sur les grammaires de Chomsky.
- ☐ Une grammaire de Chomsky peut contenir des règles unitaires.
- ☐ La forme normale de Chomsky d'une grammaire G a strictement plus de règles que G.

QCM : Question 4

À propos des analyses ascendantes qui lisent l'entrée de gauche à droite, quelles affirmations sont vraies :

- ☐ Elles reconnaissent la dérivation gauche.
- ☐ Elles reconnaissent la dérivation droite.
- ☐ On doit utiliser un automate fini pour les implanter.
- ☐ On doit utiliser un automate à pile pour les implanter.
- ☐ On doit utiliser une machine de Turing pour les implanter.

QCM : Question 5

Dans un automate à pile d'analyse ascendante,

- ☐ on décale en dépilant le prochain terminal à traiter de la pile.
- ☐ on décale en empilant le prochain terminal sur la pile.
- ☐ on réduit en lisant N terminaux d'entrée.
- ☐ on réduit en dépilant N symboles et en appliquant une règle de la grammaire dont on empile le résultat.
- ☐ il peut y avoir une séquence de transitions qui produit une analyse qui ne termine pas.

QCM : Question 6

Un parcours en profondeur :

- ☐ trouve plus rapidement une solution qu'un autre parcours.
- ☐ consomme plus de mémoire qu'un autre parcours.
- ☐ permet de ne lire l'entrée qu'une seule fois.
- ☐ a une complexité linéaire en la taille de l'entrée.

QCM : Question 7

L'algorithme de Earley :

- ☐ utilise une fonction de prédiction qui peut lire un terminal en avance.
- ☐ est exponentiel.
- ☐ ne traite pas les grammaires avec règles unitaires.
- ☐ fonctionne sur l'ensemble des grammaires hors-contextes.