UFR IM²AG

UNIVERSITÉ Grenoble Alpes

UE INF404 - Projet Logiciel

Calculatrice : étape 3

Analyse d'une expression arithmétique générale

L2 Informatique

Année 2023 - 2024

Rappel des précédents épisodes (1)

Ecrire un interpréteur d'expressions arithmétiques

Version 1 = "Expressions Arithmétiques Simples" (EAS)

- les opérandes sont des entiers
- opérateurs arithmétiques usuels (+, -, *, /)
- pas de priorités (évaluation de gauche à droite)

Exemples :

$$
\begin{aligned}
25 + 2 &\rightsquigarrow 27 \\
25 - 4 * 2 &\rightsquigarrow 42 \\
25 &\rightsquigarrow 25 \\
25 + *2 &\rightsquigarrow \text{erreur !} \\
-25 &\rightsquigarrow \text{erreur !} \\
25 \# 2 &\rightsquigarrow \text{erreur !} \\
25/0 &\rightsquigarrow \text{erreur !}
\end{aligned}
$$

Rappel des précédents épisodes (1)

Ecrire un interpréteur d'expressions arithmétiques

Version 1 = "Expressions Arithmétiques Simples" (EAS)

- les opérandes sont des entiers
- opérateurs arithmétiques usuels (+, -, *, /)
- pas de priorités (évaluation de gauche à droite)

Exemples :

$$
\begin{aligned}
25 + 2 &\rightsquigarrow 27 \\
25 - 4 * 2 &\rightsquigarrow 42 \\
25 &\rightsquigarrow 25 \\
25 + *2 &\rightsquigarrow \text{erreur !} \\
-25 &\rightsquigarrow \text{erreur !} \\
25 \# 2 &\rightsquigarrow \text{erreur !} \\
25/0 &\rightsquigarrow \text{erreur !}
\end{aligned}
$$

Rappel des précédents épisodes (1)

Ecrire un interpréteur d'expressions arithmétiques

Version 1 = "Expressions Arithmétiques Simples" (EAS)

- les opérandes sont des entiers
- opérateurs arithmétiques usuels (+, -, *, /)
- pas de priorités (évaluation de gauche à droite)

Exemples :

$$
\begin{array}{rcl}
25 + 2 & \rightsquigarrow & 27 \\
25 - 4 * 2 & \rightsquigarrow & 42 \\
25 & \rightsquigarrow & 25 \\
25 + *2 & \rightsquigarrow & \text{erreur !} \\
-25 & \rightsquigarrow & \text{erreur !} \\
25 \# 2 & \rightsquigarrow & \text{erreur !} \\
25/0 & \rightsquigarrow & \text{erreur !}
\end{array}
$$

Rappel des précédents épisodes (2)

Version 2 : EAEP

Expressions Arithmétiques Entièrement Parenthésées

$\hookrightarrow$ écrire **toute opération** entre parenthèses ...

Exemples :

$$
\begin{aligned}
25 &\rightsquigarrow 25 \\
(25 + 2) &\rightsquigarrow 27 \\
((25 - 4) * 2) &\rightsquigarrow 42 \\
(25 - (4 * 2)) &\rightsquigarrow 17 \\
(25\ \$\ 2) &\rightsquigarrow \text{erreur ! } (\textit{lexicale}) \\
25 + *2 &\rightsquigarrow \text{erreur ! } (\textit{syntaxique}) \\
25 + 4 * 2) &\rightsquigarrow \text{erreur ! } (\textit{syntaxique}) \\
25 + (4 * 2) &\rightsquigarrow \text{erreur ! } (\textit{syntaxique}) \\
(25/(5 - 5)) &\rightsquigarrow \text{erreur ! } (\textit{semantique})
\end{aligned}
$$

Rappel des précédents épisodes (2)

Version 2 : EAEP

Expressions Arithmétiques Entièrement Parenthésées

↪ écrire **toute opération** entre parenthèses ...

Exemples :

$$
\begin{array}{rcl}
25 & \rightsquigarrow & 25 \\
(25 + 2) & \rightsquigarrow & 27 \\
((25 - 4) * 2) & \rightsquigarrow & 42 \\
(25 - (4 * 2)) & \rightsquigarrow & 17 \\
(25\ \$\ 2) & \rightsquigarrow & \text{erreur ! } (\textit{lexicale}) \\
25 + *2 & \rightsquigarrow & \text{erreur ! } (\textit{syntaxique}) \\
25 + 4 * 2) & \rightsquigarrow & \text{erreur ! } (\textit{syntaxique}) \\
25 + (4 * 2) & \rightsquigarrow & \text{erreur ! } (\textit{syntaxique}) \\
(25/(5 - 5)) & \rightsquigarrow & \text{erreur ! } (\textit{semantique})
\end{array}
$$

Langage des EAEP

Lexique

- ENTIER = chiffre.(chiffre)*
- OPERATEUR = '+' + '-' + '*' + '/'
- PARO = '(' et PARF = ')'

Syntaxe

langage non régulier

Ne peut être décrite par un automate ⇒ grammaire

$$
\begin{aligned}
exp &\rightarrow eaep\ \texttt{FIN\_SEQUENCE} \\
eaep &\rightarrow \texttt{ENTIER} \\
eaep &\rightarrow \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF} \\
op &\rightarrow \texttt{PLUS} \\
op &\rightarrow \texttt{MOINS} \\
op &\rightarrow \texttt{MUL}
\end{aligned}
$$

Langage des EAEP

Lexique

- ENTIER = chiffre.(chiffre)*
- OPERATEUR = '+' + '-' + '*' + '/'
- PARO = '(' et PARF = ')'

Syntaxe

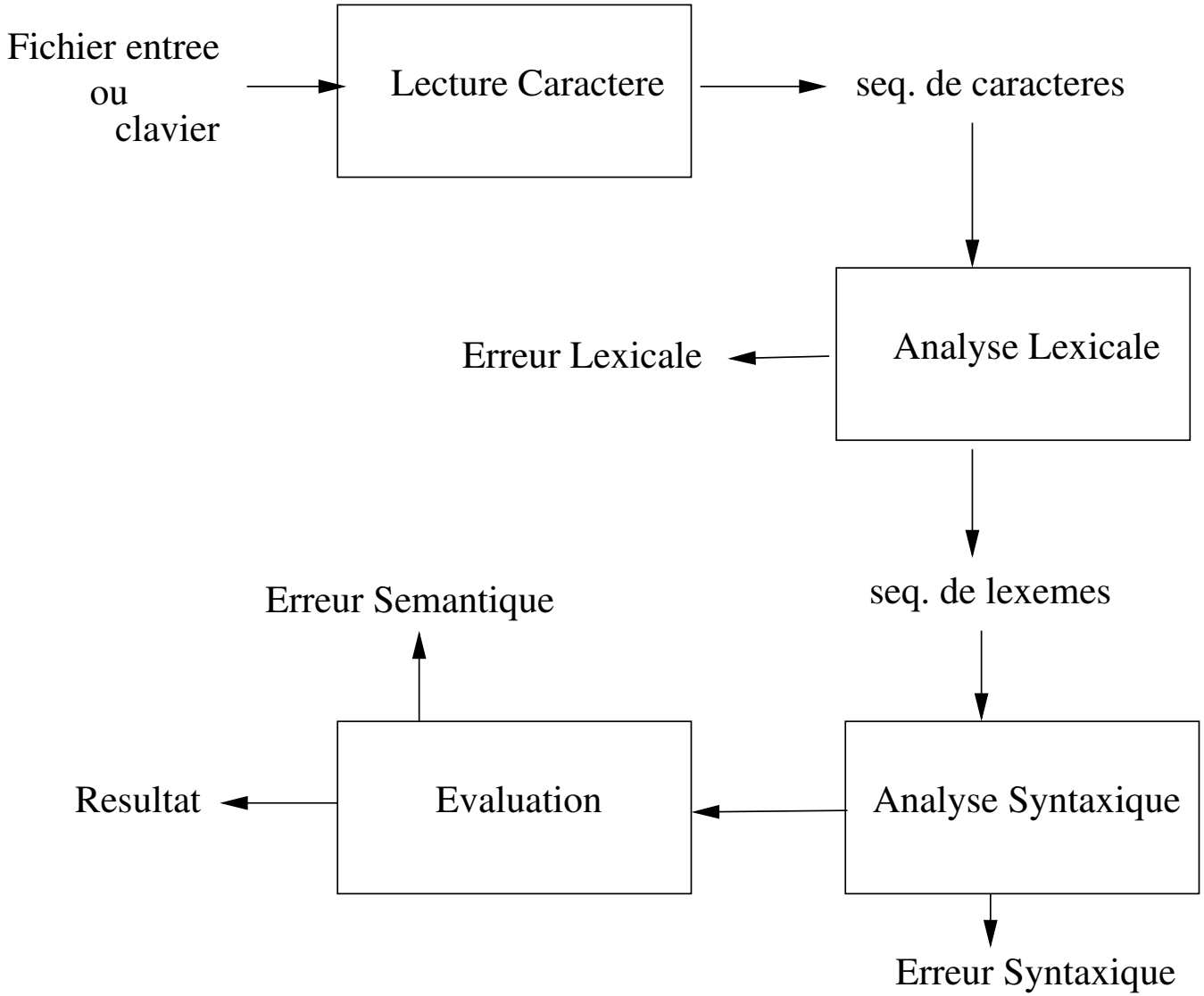
langage non régulier

Ne peut être décrite par un automate ⇒ grammaire

$$
\begin{aligned}
exp &\rightarrow \textit{eaep}\ \texttt{FIN\_SEQUENCE} \\
eaep &\rightarrow \texttt{ENTIER} \\
eaep &\rightarrow \texttt{PARO}\ \textit{eaep}\ \textit{op}\ \textit{eaep}\ \texttt{PARF} \\
op &\rightarrow \texttt{PLUS} \\
op &\rightarrow \texttt{MOINS} \\
op &\rightarrow \texttt{MUL}
\end{aligned}
$$

Structure de la calculatrice

Quatre composants/modules principaux . . .



La suite ? . . .Expressions Arithmètiques Générales (EAG)

Généralisation : priorités, donc parenthèses . . .

ex : $5 + 3 * 4 = 17 \quad (5 + 3) * 4 = 32$

Nouveau langage

- modifier la syntaxe :
  régle de "bonne imbrication des parenthèses"
  . . .mais parenthèses non systématiques . . . !
  $\Rightarrow$ définir une grammaire des EAG ?
- modifier la sémantique :
  règles de calcul avec priorité des opérateurs

Conséquences en terme de programmation ?

1. analyse syntaxique : à partir de la grammaire ?
2. évaluation ?

La suite ? . . .Expressions Arithmètiques Générales (EAG)

Généralisation : priorités, donc parenthèses . . .

ex : $5 + 3 * 4 = 17 \quad (5 + 3) * 4 = 32$

Nouveau langage

- modifier la syntaxe :
  régle de "bonne imbrication des parenthèses"
  . . .mais parenthèses non systématiques . . . !
  $\Rightarrow$ définir une grammaire des EAG ?
- modifier la sémantique :
  règles de calcul avec priorité des opérateurs

Conséquences en terme de programmation ?

1. analyse syntaxique : à partir de la grammaire ?
2. évaluation ?

La suite ? . . .Expressions Arithmètiques Générales (EAG)

Généralisation : priorités, donc parenthèses . . .

ex : $5 + 3 * 4 = 17 \quad (5 + 3) * 4 = 32$

Nouveau langage

- modifier la syntaxe :
  régle de "bonne imbrication des parenthèses"
  . . .mais parenthèses non systématiques . . . !
  $\Rightarrow$ définir une grammaire des EAG ?
- modifier la sémantique :
  règles de calcul avec priorité des opérateurs

Conséquences en terme de programmation ?

1. analyse syntaxique : à partir de la grammaire ?
2. évaluation ?

Retour sur la grammaire des EAEP

$$
\begin{aligned}
\textit{exp} &\rightarrow \textit{eaep}\ \texttt{FIN\_SEQUENCE} \\
\textit{eaep} &\rightarrow \texttt{ENTIER} \\
\textit{eaep} &\rightarrow \texttt{PARO}\ \textit{eaep}\ \textit{op}\ \textit{eaep}\ \texttt{PARF} \\
\textit{op} &\rightarrow \texttt{PLUS} \\
\textit{op} &\rightarrow \texttt{MOINS} \\
\textit{op} &\rightarrow \texttt{MUL}
\end{aligned}
$$

- $\{\texttt{FIN\_SEQUENCE}, \texttt{ENTIER}, \texttt{PARO}, \texttt{PARF}, \texttt{PLUS}, \textit{etc}\} =$ lexèmes du langage = vocabulaire **terminal** de la grammaire
- $\{\textit{exp}, \textit{op}, \textit{eaep}\}$ = vocabulaire **non-terminal** de la grammaire
- $\textit{exp}$ = **axiome** de la grammaire
- Six **règles** dans cette grammaire ...

Grammaire (Hors-Contexte) : définition

Quadruplet $G = (V_t, V_n, z, P)$ tel que :

- V_t : vocabulaire terminal (les lexèmes)
  ensemble fini de symboles, notés en majuscules
- V_n : vocabulaire non-terminal
  ensemble fini de symboles, notés en minuscules ($V_t \cap V_n = \emptyset$)
- $Z \in V_n$: axiome
- P : ensemble des règles (ou productions)
  $P \subseteq V_n \times (V_t \cup V_n)^*$
  Les éléments (X, α) de P seront notés $X \to \alpha$.

Reconnaitre une EAEP : exemples de dérivations

$$
\begin{array}{rcl}
exp & \rightarrow & eaep\ \texttt{FIN\_SEQUENCE} \\
eaep & \rightarrow & \texttt{ENTIER} \\
eaep & \rightarrow & \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF} \\
op & \rightarrow & \texttt{PLUS} \\
op & \rightarrow & \texttt{MOINS} \\
op & \rightarrow & \texttt{MUL}
\end{array}
$$

$$
\begin{array}{rcl}
exp & \Rightarrow_G & eaep\ \texttt{FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{ENTIER FIN\_SEQUENCE}
\end{array}
$$

$$
\begin{array}{rcl}
exp & \Rightarrow_G & eaep\ \texttt{FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER}\ op\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER PLUS}\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER PLUS ENTIER PARF FIN\_SEQUENCE}
\end{array}
$$

Reconnaitre une EAEP : exemples de dérivations

$$
\begin{array}{rcl}
exp & \rightarrow & eaep\ \texttt{FIN\_SEQUENCE} \\
eaep & \rightarrow & \texttt{ENTIER} \\
eaep & \rightarrow & \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF} \\
op & \rightarrow & \texttt{PLUS} \\
op & \rightarrow & \texttt{MOINS} \\
op & \rightarrow & \texttt{MUL}
\end{array}
$$

$$
\begin{array}{rcl}
exp & \Rightarrow_G & eaep\ \texttt{FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{ENTIER FIN\_SEQUENCE}
\end{array}
$$

$$
\begin{array}{rcl}
exp & \Rightarrow_G & eaep\ \texttt{FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER}\ op\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER PLUS}\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER PLUS ENTIER PARF FIN\_SEQUENCE}
\end{array}
$$

Reconnaitre une EAEP : exemples de dérivations

$$
\begin{array}{rcl}
exp & \rightarrow & eaep\ \texttt{FIN\_SEQUENCE} \\
eaep & \rightarrow & \texttt{ENTIER} \\
eaep & \rightarrow & \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF} \\
op & \rightarrow & \texttt{PLUS} \\
op & \rightarrow & \texttt{MOINS} \\
op & \rightarrow & \texttt{MUL}
\end{array}
$$

$$
\begin{array}{rcl}
exp & \Rightarrow_G & eaep\ \texttt{FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{ENTIER FIN\_SEQUENCE}
\end{array}
$$

$$
\begin{array}{rcl}
exp & \Rightarrow_G & eaep\ \texttt{FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO}\ eaep\ op\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER}\ op\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER PLUS}\ eaep\ \texttt{PARF FIN\_SEQUENCE} \\
 & \Rightarrow_G & \texttt{PARO ENTIER PLUS ENTIER PARF FIN\_SEQUENCE}
\end{array}
$$

Dérivations

$G = (V_t, V_n, Z, P)$ une grammaire hors-contexte, $w, w' \in (V_t \cup V_n)^*$.

- w' est un dérivé direct par G de w, noté $w \Rightarrow_G w'$ ssi :
  il existe $X \in V_n$, il existe $u, t, v \in (V_t \cup V_n)^*$ tels que

$$w = uXv \text{ et } w' = utv \text{ et } X \to t \in P.$$

- w' est un dérivé par G de w, noté $w \Rightarrow_G^* w'$, ssi :
  il existe $w_0, w_1, \ldots, w_n \in (V_t \cup V_n)^*$ tels que

$$w_0 = w, w_n = w' \text{ et pour tout i } w_{i+1} \text{ est un dérivé direct de } w_i.$$

Remarque : $\Rightarrow_G^*$ est la fermeture transitive de $\Rightarrow_G$.

Langages hors-contextes

$G = (V_t, V_n, Z, P)$ une grammaire hors-contexte.
Le langage défini par G (ou reconnu par G) est l'ensemble $L(G)$ tel que :

$$L(G) = \{w \in V_t^* \mid Z \Rightarrow_G^* w\}$$

Un langage L est dit **hors-contexte** s'il existe une grammaire hors-contexte qui le reconnaît.

Remarque : Tout langage régulier est un langage hors-contexte, il existe des langages hors-contexte qui ne sont pas réguliers.

Langages hors-contextes

$G = (V_t, V_n, Z, P)$ une grammaire hors-contexte.
Le langage défini par G (ou reconnu par G) est l'ensemble $L(G)$ tel que :

$$L(G) = \{w \in V_t^* \mid Z \Rightarrow_G^* w\}$$

Un langage L est dit **hors-contexte** s'il existe une grammaire hors-contexte qui le reconnaît.

Remarque : Tout langage régulier est un langage hors-contexte, il existe des langages hors-contexte qui ne sont pas réguliers.

Langages hors-contextes

$G = (V_t, V_n, Z, P)$ une grammaire hors-contexte.
Le langage défini par G (ou reconnu par G) est l'ensemble $L(G)$ tel que :

$$L(G) = \{w \in V_t^* \mid Z \Rightarrow_G^* w\}$$

Un langage L est dit **hors-contexte** s'il existe une grammaire hors-contexte qui le reconnaît.

Remarque : Tout langage régulier est un langage hors-contexte, il existe des langages hors-contexte qui ne sont pas réguliers.

Analyse Syntaxique

Soit $G = (V_t, V_n, Z, P)$ une grammaire hors-contexte,
soit ω un séquence de V_t^*.

Analyse syntaxique = déterminer si $\omega \in L(G)$?

Contraintes pratiques :

- algorithme **linéaire** en fonction de $|\ \omega\ |$
- accés **séquentiel** à ω, de gauche à droite

$\Rightarrow$ possible uniquement pour un **sous-ensemble strict** des langages hors-contexte

Problèmes posés :

- Est-ce que le **langage** considéré appartient à cette classe ?
- Si oui, comment construire (efficacement) un analyseur (efficace) ?

$\Rightarrow$ réponse en fonction de la **structure** de la **grammaire** considérée !

Analyse Syntaxique

Soit $G = (V_t, V_n, Z, P)$ une grammaire hors-contexte,
soit ω un séquence de V_t^*.

Analyse syntaxique = déterminer si $\omega \in L(G)$?

Contraintes pratiques :

- algorithme **linéaire** en fonction de $|\ \omega\ |$
- accés **séquentiel** à ω, de gauche à droite

$\Rightarrow$ possible uniquement pour un **sous-ensemble strict** des langages hors-contexte

Problèmes posés :

- Est-ce que le **langage** considéré appartient à cette classe ?
- Si oui, comment construire (efficacement) un analyseur (efficace) ?

$\Rightarrow$ réponse en fonction de la **structure** de la **grammaire** considérée !

Analyse Syntaxique

Soit $G = (V_t, V_n, Z, P)$ une grammaire hors-contexte,
soit ω un séquence de V_t^*.

Analyse syntaxique = déterminer si $\omega \in L(G)$?

Contraintes pratiques :

- algorithme **linéaire** en fonction de $|\ \omega\ |$
- accés **séquentiel** à ω, de gauche à droite

$\Rightarrow$ possible uniquement pour un **sous-ensemble strict** des langages hors-contexte

Problèmes posés :

- Est-ce que le **langage** considéré appartient à cette classe ?
- Si oui, comment construire (efficacement) un analyseur (efficace) ?

$\Rightarrow$ réponse en fonction de la **structure** de la **grammaire** considérée !

Dans la suite ...

- On considère une technique d'analyse dite **descendante** :
  - produire ω à partir de Z
  - produire une dérivation gauche $Z \Longrightarrow_G^* \omega$

- Cette technique se programme **directement à partir de la grammaire** par un ensemble de **procédures mutuellement récursives** (c.f "eaep") :
  A chaque non-terminal $X \rightsquigarrow$ une procédure `Rec_X`

- On doit donc :
  - Ecrire la grammaire de notre langage sous une forme qui facilite l'analyse descendante (**grammaire LL(1)**)
  - programmer l'**analyseur syntaxique**, à partir de la grammaire

Dans la suite ...

- On considère une technique d'analyse dite **descendante** :
  - produire ω à partir de Z
  - produire une dérivation gauche $Z \Longrightarrow_G^* \omega$

- Cette technique se programme **directement à partir de la grammaire** par un ensemble de **procédures mutuellement récursives** (c.f "eaep") :
  A chaque non-terminal $X \rightsquigarrow$ une procédure `Rec_X`

- On doit donc :
  - Ecrire la grammaire de notre langage sous une forme qui facilite l'analyse descendante (**grammaire LL(1)**)
  - programmer l'**analyseur syntaxique**, à partir de la grammaire

Ecrire un analyseur à partir de la grammaire ?

Exemple des EAEP

eaep → ENTIER

eaep → PARO *eaep op eaep* PARF

```
Rec_eaep =
   selon LC.nature
        cas ENTIER : Avancer
        cas PARO : Avancer ; Rec_eaep ; Rec_op ; Rec_eap ;
           si LC.nature = PARF alors Avancer sinon Erreur
        autre : Erreur
fin
```

op → PLUS

op → MOINS

op → MUL

```
Rec_op =
   selon LC.nature
        cas PLUS, MUL, MOINS : Avancer
        autre : Erreur
fin
```

Ecrire un analyseur à partir de la grammaire ?

Exemple des EAEP

$$\begin{aligned} \textit{eaep} &\rightarrow \texttt{ENTIER} \\ \textit{eaep} &\rightarrow \texttt{PARO}\ \textit{eaep}\ \textit{op}\ \textit{eaep}\ \texttt{PARF} \end{aligned}$$

```
Rec_eaep =
   selon LC.nature
        cas ENTIER : Avancer
        cas PARO : Avancer ; Rec_eaep ; Rec_op ; Rec_eap ;
           si LC.nature = PARF alors Avancer sinon Erreur
        autre : Erreur
fin
```

$$\begin{aligned} \textit{op} &\rightarrow \texttt{PLUS} \\ \textit{op} &\rightarrow \texttt{MOINS} \\ \textit{op} &\rightarrow \texttt{MUL} \end{aligned}$$

```
Rec_op =
   selon LC.nature
        cas PLUS, MUL, MOINS : Avancer
        autre : Erreur
fin
```

Expressions Arithmétiques Générales (EAG)

$\rightarrow$ parenthèses non obligatoires, priorités entre les opérateurs

Exemples : `5` ; `5 + 3` ; `5 + 3 * 2` ; `(5+3) * 2` ; etc.

Grammaire 1

$$
\begin{array}{rcl}
\textit{eag} & \rightarrow & \texttt{ENTIER} \\
\textit{eag} & \rightarrow & \textit{eag}\ \textit{op}\ \textit{eag} \\
\textit{eag} & \rightarrow & \texttt{PARO}\ \textit{eag}\ \texttt{PARF} \\
\textit{op} & \rightarrow & \texttt{PLUS} \\
\textit{op} & \rightarrow & \texttt{MOINS} \\
\textit{op} & \rightarrow & \texttt{MUL}
\end{array}
$$

Problèmes

- priorités des opérateurs ?
- écriture de l'analyseur ? récursion infinie ?

Modifier la grammaire des EAG ? (1)

$\rightarrow$ prendre en compte les priorités $\Rightarrow$ structurer la grammaire [1]

Grammaire 2

eag	$\rightarrow$	*seq_terme*
seq_terme	$\rightarrow$	*seq_terme op1 terme*
seq_terme	$\rightarrow$	*terme*
terme	$\rightarrow$	*seq_facteur*
seq_facteur	$\rightarrow$	*seq_facteur op2 facteur*
seq_facteur	$\rightarrow$	*facteur*
facteur	$\rightarrow$	*ENTIER*
facteur	$\rightarrow$	`PARO` *eag* `PARF`
op1	$\rightarrow$	`PLUS`
op1	$\rightarrow$	`MOINS`
op2	$\rightarrow$	`MUL`

1. cf. prochain cours !

Modifier la grammaire des EAG ? (2)

$\rightarrow$ éliminer la "récursivité à gauche"

Grammaire 3

$$
\begin{array}{rcl}
\mathit{eag} & \rightarrow & \mathit{seq\_terme} \\
\mathit{seq\_terme} & \rightarrow & \mathit{terme}\ \mathit{suite\_seq\_terme} \\
\mathit{suite\_seq\_terme} & \rightarrow & \mathit{op1}\ \mathit{terme}\ \mathit{suite\_seq\_terme} \\
\mathit{suite\_seq\_terme} & \rightarrow & \varepsilon \\
\mathit{terme} & \rightarrow & \mathit{seq\_facteur} \\
\mathit{seq\_facteur} & \rightarrow & \mathit{facteur}\ \mathit{suite\_seq\_facteur} \\
\mathit{suite\_seq\_facteur} & \rightarrow & \mathit{op2}\ \mathit{facteur}\ \mathit{suite\_seq\_facteur} \\
\mathit{suite\_seq\_facteur} & \rightarrow & \varepsilon \\
\mathit{facteur} & \rightarrow & \mathit{ENTIER} \\
\mathit{facteur} & \rightarrow & \texttt{PARO}\ \mathit{eag}\ \texttt{PARF} \\
\mathit{op1} & \rightarrow & \texttt{PLUS} \\
\mathit{op1} & \rightarrow & \texttt{MOINS} \\
\mathit{op2} & \rightarrow & \texttt{MUL}
\end{array}
$$

Vérifier la syntaxe d'une EAG (1)

→ un algo récursif, **basé sur la grammaire**

$$eag \rightarrow seq\_terme$$

```
Rec_eag() =
        Rec_seq_terme() ;
```

$$seq\_terme \rightarrow terme\ suite\_seq\_terme$$

```
Rec_seq_terme() =
        Rec_terme() ; Rec_suite_seq_terme() ;
```

Vérifier la syntaxe d'une EAG (1)

→ un algo récursif, **basé sur la grammaire**

$$eag \rightarrow seq\_terme$$

```
Rec_eag() =
          Rec_seq_terme() ;
```

$$seq\_terme \rightarrow terme\ suite\_seq\_terme$$

```
Rec_seq_terme() =
           Rec_terme() ; Rec_suite_seq_terme() ;
```

Vérifier la syntaxe d'une EAG (2)

lexèmes **directeurs** : indiquent la règle à dériver !

$$\textit{suite\_seq\_terme} \rightarrow \textit{op}1\ \textit{terme}\ \textit{suite\_seq\_terme} \qquad [\texttt{PLUS}, \texttt{MOINS}]$$

$$\textit{suite\_seq\_terme} \rightarrow \varepsilon$$

```
Rec_suite_seq_terme() =
   selon LC().nature  // LC est le lexeme_courant()
      cas PLUS, MOINS :
               Avancer() ; Rec_terme() ; Rec_suite_seq_terme()
      autre : // Rien, epsilon !
fin
```

Idem pour `Rec_seq_facteur()` et `Rec_suite_seq_facteur()` ...

Vérifier la syntaxe d'une EAG (2)

lexèmes **directeurs** : indiquent la règle à dériver !

$$\textit{suite\_seq\_terme} \rightarrow \textit{op1}\ \textit{terme}\ \textit{suite\_seq\_terme} \qquad [\texttt{PLUS}, \texttt{MOINS}]$$

$$\textit{suite\_seq\_terme} \rightarrow \varepsilon$$

```
Rec_suite_seq_terme() =
    selon LC().nature  // LC est le lexeme_courant()
       cas PLUS, MOINS :
                Avancer() ; Rec_terme() ; Rec_suite_seq_terme()
       autre : // Rien, epsilon !
fin
```

Idem pour `Rec_seq_facteur()` et `Rec_suite_seq_facteur()` ...

Vérifier la syntaxe d'une EAG (2)

lexèmes **directeurs** : indiquent la règle à dériver !

$$\textit{suite\_seq\_terme} \rightarrow \textit{op}1\ \textit{terme}\ \textit{suite\_seq\_terme} \qquad [\texttt{PLUS}, \texttt{MOINS}]$$

$$\textit{suite\_seq\_terme} \rightarrow \varepsilon$$

```
Rec_suite_seq_terme() =
    selon LC().nature  // LC est le lexeme_courant()
       cas PLUS, MOINS :
                Avancer() ; Rec_terme() ; Rec_suite_seq_terme()
       autre : // Rien, epsilon !
fin
```

Idem pour `Rec_seq_facteur()` et `Rec_suite_seq_facteur()` ...

Vérifier la syntaxe d'une EAG (3)

$$\textit{facteur} \rightarrow \textit{ENTIER}$$

$$\textit{facteur} \rightarrow \texttt{PARO}\ \textit{eag}\ \texttt{PARF}$$

```
Rec_facteur() =
   selon LC().nature  // LC est le lexeme_courant()
        cas ENTIER : Avancer()
        cas PARO : Avancer() ; Rec_eag() ;
                   si LC.nature = PARF alors Avancer sinon Erreur
        autre : Erreur
fin
```

$$\textit{op1} \rightarrow \texttt{PLUS}$$

$$\textit{op1} \rightarrow \texttt{MOINS}$$

```
Rec_op1 =
   selon LC.nature
        cas PLUS, MOINS : Avancer
        autre : Erreur
fin
```

Idem pour Rec_op2 . . .

Vérifier la syntaxe d'une EAG (3)

$$\textit{facteur} \rightarrow \textit{ENTIER}$$

$$\textit{facteur} \rightarrow \texttt{PARO}\ \textit{eag}\ \texttt{PARF}$$

```
Rec_facteur() =
   selon LC().nature  // LC est le lexeme_courant()
        cas ENTIER : Avancer()
        cas PARO : Avancer() ; Rec_eag() ;
                   si LC.nature = PARF alors Avancer sinon Erreur
        autre : Erreur
fin
```

$$\textit{op1} \rightarrow \texttt{PLUS}$$

$$\textit{op1} \rightarrow \texttt{MOINS}$$

```
Rec_op1 =
   selon LC.nature
        cas PLUS, MOINS : Avancer
        autre : Erreur
fin
```

Idem pour Rec_op2 . . .

Dans la suite (les prochains TPs) . . .

1. Ecrire l'analyseur syntaxique pour le langage *eag* à partir de la **Grammaire 3**.

2. Etendre cet analyseur pour calculer la valeur de l'expression (après le prochain cours !)