Rappel des précédents épisodes

Ecrire un *interpréteur d'expressions arithmétiques*
(∼ "calculatrice en ligne", comme la commande `bc` sous Linux)

Version initiale = "Expressions Arithmétiques Simples" (EAS)

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

Exemples :

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

Etape 1 : spécifier le langage d'entrée (1)

Alphabet = ensemble des caractères autorisés

V = {0, 1, 2, ..., 9, +, -, *, /, *espace*, *tabulation*, *fin-de-ligne*}
On pourra ajouter :

- des lettres (pour écrire des opérateurs `plus`, `moins`, `exp`, `log`, etc.)
- le caractère '.' (pour érire des "nombres à virgules")
- les parenthèses ouvrantes et fermantes
- etc.

Les caractères *espace*, *tabulation*, *fin-de-ligne* sont des **séparateurs**

Lexique = ensemble des "mots" du langage

Deux classes de lexèmes :

- entiers : séquence non vide de chiffres
- opérateurs : PLUS ('+'), MOINS ('-'), MULT ('*'), DIV ('/')

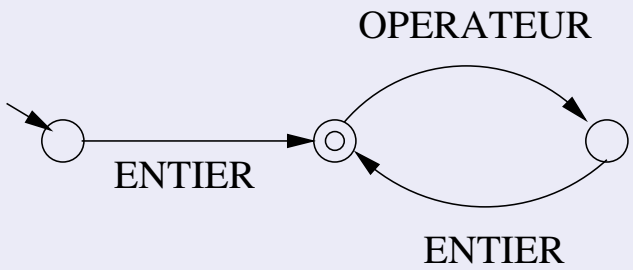
→ peuvent être définis par un **automate** ou une **expression régulière**

Etape 1 : spécifier le langage d'entrée (2)

Syntaxe = ensemble des "phrases bien formées"

Expression régulière (ou automate) sur les lexèmes :

$$\text{entier}.(\text{operateur}.\text{entier})^*$$

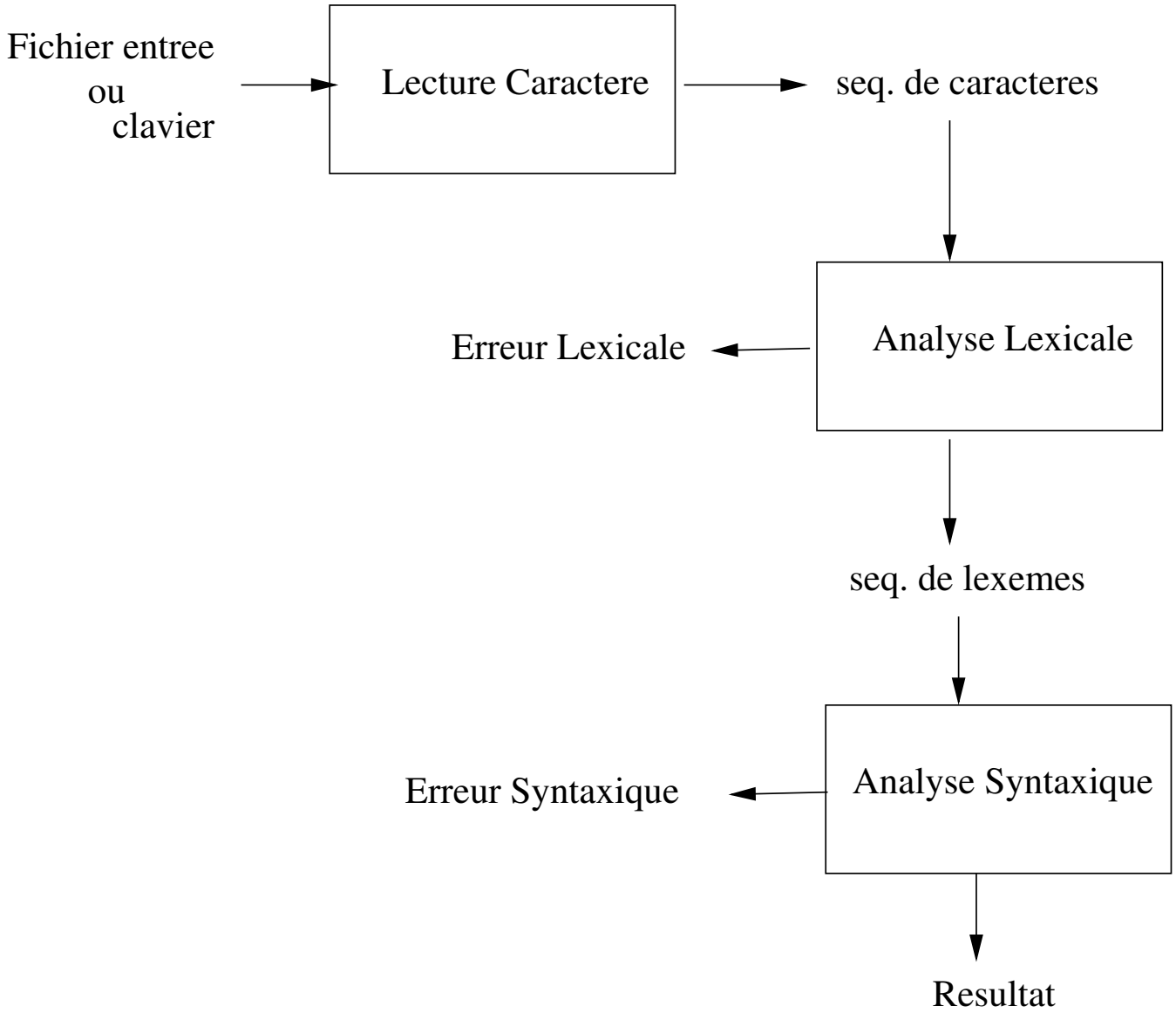


Exo : ajouter le "moins unaire" ?

Sémantique = ensemble des phrases "qui ont un sens"

règles de l'arithmétique (pas de division par 0 !)

Etape 2 : Structure de l'interpréteur



Modules (c.f. TP1)

Lecture_Caractères (*)

Accès à une *séquence de caractères*

Primitives : `demarrer_car`, `avancer_car`, `caractere_courant`, `fin_de_sequence_car`

Analyse_Lexicale (*)

Accès à une *séquence de lexèmes*

Primitives : `demarrer`, `avancer`, `lexeme_courant`, `fin_de_sequence`

Analyse_Syntaxique

vérifie la syntaxe . . .et calcule le résultat !
(l'algo de calcul du résultat se déduit de l'automate)

Primitives : `analyser`

→ **Un seul parcours (gauche-droite) du fichier d'entrée . . .**

(*) fournis, à compléter/modifier le cas échéant . . .

La suite ?

Etendre cette version

- nombres à virgules ($25.2 - 7.36$)
- nouveaux opérateurs (`exp`, `modulo`, `plus`, etc.)
- "moins unaire" ($-25 + 12$, $- - -25 + - - 12$)

Généralisation : priorités, donc parenthèses ...

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

- nouveaux lexèmes : `PARO` et `PARF`
  $\rightarrow$ on peut étendre l'analyse lexicale ...

Mais :

- la syntaxe ne se décrit plus par un automate
  pas un **langage régulier** (imbrication de parenthèses)
- algo d'analyse et d'évaluation ? ? ?

$\Rightarrow$ **définir un nouveau formalisme ?**

Etape intermédiaire : les EAEP

EAEP = Expressions Arithmétiques Entièrement Parenthésées

Idée = écrire **toute opération** entre parenthèses . . .

Exemples :

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

Définir une eaep ?

L'ensemble *eaep* est défini par les règles suivantes :

1. `ENTIER` est une *eaep*
2. `PARO` eaep `PLUS` eaep `PARF` est une *eaep*
3. `PARO` eaep `MOINS` eaep `PARF` est une *eaep*
4. `PARO` eaep `MUL` eaep `PARF` est une *eaep*
5. `PARO` eaep `DIV` eaep `PARF` est une *eaep*

$\rightarrow$ définition **récursive** des *eaep*
(schéma inductif, $\sim$ schéma de récurrence)

Autre écriture : grammaire des *eaep*

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

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

Vérifier la syntaxe d'une *eaep*

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

---

eaep → ENTIER

eaep → PARO *eaep op eaep* PARF

```
Rec_eaep =
   selon LC.nature
        cas ENTIER : Avancer
        cas PARO : Avancer ; Rec_eaep ; Rec_op ; Rec_eaep ;
           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
```

Conclusion (provisoire !)

- un formalisme pour décrire la syntaxe d'un langage non régulier

  $\rightarrow$ les **grammaires**

- on peut **déduire** de ce formalisme un algorithme d'**analyse syntaxique** (toujours ? ? ?)

$\Rightarrow$ Mise en oeuvre au TP2, et généralisation aux cours suivants !