

|   |                |  |                |
|---|----------------|--|----------------|
| Traitement informatique des langages informatiques          | 1 Introduction | Traitement informatique des langages informatiques | 1 Introduction |
| <h1>Traitement informatique des langages informatiques</h1> |                |  |                |
| Sylvain.Boulme@univ-grenoble-alpes.fr                       |                | 1.1 Contexte du cours                              |                |
| Diaporama & notes du cours GC période 2                     |                | 1.2 Introduction aux concepts de base              |                |

|  |      |  |      |
|--|------|--|------|
| Traitement informatique des langages informatiques   | 1/16 | 1.1 Contexte du cours  | 2/16 |
| <h2>“Traitement informatique des langages informatiques” ?</h2>  |      |  |      |
| NB : langage informatique = conçu pour rendre possible certains traitements informatiques (par opposition aux langues naturelles). |      | Travailler en parallèle CTD (théorie) et TP (pratique)   |      |
| <b>Objectifs pédagogiques du cours</b>   |      | <ul style="list-style-type: none"> <li>▶ Le sujet du TP reformule (presque) entièrement le cours en articulant les différentes notions sur un cas concret.</li> <li>▶ Le TP commence directement par les applications du chapitre 3 (donc en avance de +seulement quelques semaines sur le cours).</li> <li>▶ Le CTD fait dès le chapitre 2 référence aux exemples du TP.</li> <li>▶ En début de période 3, le CTD aide à réaliser le TP.</li> </ul> |      |

|   |  |
|---|--|
| <p>Traitement informatique des langages informatiques</p> <h2>Liens avec les autres cours</h2> <ul style="list-style-type: none"> <li>▶ Automates finis (langages réguliers, automates).</li> <li>▶ Projet GL (compilation d'un mini-Java).</li> <li>▶ Logique (logique propositionnelle et notion de termes).</li> <li>▶ Algorithmiques (notamment algos sur les arbres).</li> <li>▶ Logiciel de base (trad. "haut niveau" vers "bas niveau").</li> <li>▶ en 2A : Sémantique &amp; analyse de programme.</li> <li>▶ en 2A : ACVL (patrons de conception).</li> </ul> | <p>1 Introduction</p> <p>Traitement informatique des langages informatiques</p> <h2>Ouvrages conseillés pour approfondir le cours</h2> <ul style="list-style-type: none"> <li><a href="#">Compilers: Principles, Techniques and Tools</a><br/>de Aho, Lam, Sethi &amp; Ullman (1988/2007)</li> <li><a href="#">Introduction to automata theory, languages, and computation</a><br/>de Hopcroft, Motwani &amp; Ullman (1979/2007)</li> <li><a href="#">The Definitive ANTLR4 Reference</a><br/>de Terence Parr (2013)</li> <li><a href="#">Theories of programming languages</a><br/>de John C. Reynolds (1998)</li> </ul> <p>NB: livres tous disponibles dans BUs de Grenoble.<br/>cf. <a href="http://rugbis.grenet.fr">http://rugbis.grenet.fr</a></p> |
|---|--|

|  |   |
|--|---|
| <p>1.1 Contexte du cours</p> <p>Traitement informatique des langages informatiques</p> <h2>Chapitre 1 Introduction</h2> <ul style="list-style-type: none"> <li><a href="#">1.1 Contexte du cours</a></li> <li><a href="#">1.2 Introduction aux concepts de base</a></li> </ul> | <p>5/16</p> <p>1 Introduction</p> <p>1.1 Contexte du cours</p> <p>Traitement informatique des langages informatiques</p> <h2>Notion d'interpréteur</h2> <p><b>Définition</b> Interpréteur = programme qui prend en entrée un texte (dans un certain "<i>langage source</i>") et effectue un certain traitement spécifié par ce texte.</p> <p><b>Exemples</b></p> <ul style="list-style-type: none"> <li>▶ machine virtuelle java (interpréteur de fichiers ".class")</li> <li>▶ compilateur java (traduit du ".java" en ".class")</li> <li>▶ interpréteurs bash, perl, python, etc.</li> <li>▶ programme "grep" (interpréteur d'expressions régulières).</li> <li>▶ pdflatex (traduit du latex vers pdf).</li> <li>▶ visualiseurs "pdf" (evince, acrobat-reader).</li> <li>▶ navigateur web (interprète html+javascript).</li> <li>▶ ...</li> </ul> |
|--|---|

## Méta-interpréteurs

**Déf** Méta-interpréteur = pg pour construire des interpréteurs.

En entrée : spécification *haut-niveau* d'un interpréteur.

En sortie : génère un programme qui réalise cet interpréteur.

### Exemples

- ▶ YACC "Yet Another Compiler Compiler"  
(outil historique depuis 70s)
- ▶ BISON (raffinement de Yacc)  
outil GNU qui produit du C, C++ ou Java.  
longtemps utilisé dans implémentation de gcc.  
démonstration au chapitre 4.
- ▶ ANTLR "ANother Tool for Language Recognition"  
produit Java ou C#, utilisé en ProjetGL.

## Syntaxe versus Sémantique

**Syntaxe** = codage pour "*manipuler*" (communiquer, raisonner, calculer, etc)

*Exemple de plusieurs syntaxes d'une expression régulière*

1. *notation mathématique* (Kleene – 1956)  
 $(a + d)^* . b$
2. *syntaxe POSIX BRE* (ISO/IEC 9945-2 :1993)  
 $\backslash(a|d\)*b$

**Sémantique** = sens = signification

*Expression régulière ci-dessus représente l'ensemble infini de mots*

$$\{b, ab, db, aab, adb, dab, ddb, \dots\}$$

c.-à-d. le plus petit langage  $X$  qui satisfait l'équation

$$X = \{b\} \cup \{a, d\} \cdot X$$

## Structure d'un interpréteur

### Traitement d'une "phrase" d'entrée

#### 1. Analyse syntaxique (parsing)

lecture + conversion de la suite des caractères dans une SdD interne appelée *yntaxe abstraite*.

#### 2. Éventuellement étapes internes de compilation

vérifications & traductions dans structures de données internes.

#### 3. Exécution du traitement (backend)

à partir des SdD internes.

### Batch (tâche de fond) / interactif

| Entrée     |  |
|------------|--|
| Batch      | éventuellement lue comme <i>une seule grosse phrase</i> .  |
| Interactif | découpée en <i>suite de phrases traitées en séquence</i> . |

## Syntaxe textuelle versus syntaxe abstraite

**Syntaxe textuelle** = suite des caractères.

*synonyme : syntaxe concrète.*

- ▶ "(x+y) . z"
- ▶ "(( (x) + (y) ) . (z))"
- ▶ "x+(y.z)"
- ▶ "x+y.z"

**Syntaxe abstraite** = arbre = structure du parenthésage.  
*arbre abstrait, "Abstract Syntax Tree" (AST) en anglais.*



NB : *structure grammaticale* d'une phrase en langue naturelle.

**Analyse syntaxique** = construire AST depuis suite de caractères.

## Langages informatiques versus langues naturelles

### Langages informatiques

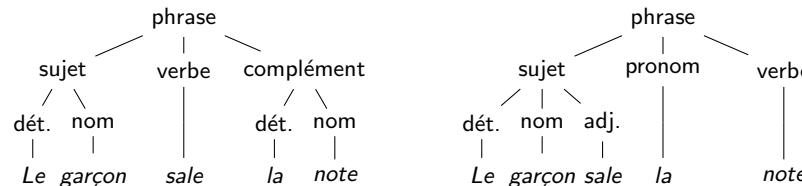
Analyse syntaxique indépendante de la sémantique !

Objectifs : traitements automatiques & modulaires & sans ambiguïtés.

### Langues humaines

Interactions entre analyse syntaxique & analyse sémantique

Exemple : "Le garçon sale la note"



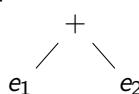
⇒ traitement automatique imparfait des langues naturelles !  
cf. correcteurs orthographiques, traducteurs automatiques, etc.

## Exemple : syntaxe abstraite des expr. régulières sur $V^*$

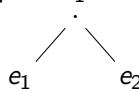
Soit  $V$  ensemble fini.

**RegExp** = type d'ASTs défini par 6 types de noeuds :

- ▶ constantes  $\epsilon$  ou  $\emptyset$  ou lettre  $a$  (pour  $a \in V$ )
- ▶ pour  $e_1$  et  $e_2$  de **RegExp**,



- ▶ pour  $e_1$  et  $e_2$  de **RegExp**,



- ▶ pour  $e$  de **RegExp**



## Exemple : sémantique des expr. régulières sur $V^*$

La sémantique de  $e$  notée  $\llbracket e \rrbracket \in \mathcal{P}(V^*)$  (= langage reconnu par  $e$ ) est définie par cas sur chaque type de noeuds :

$$\begin{aligned} \llbracket \emptyset \rrbracket &\stackrel{\text{def}}{=} \emptyset \\ \llbracket \epsilon \rrbracket &\stackrel{\text{def}}{=} \{\epsilon\} \\ \llbracket a \rrbracket &\stackrel{\text{def}}{=} \{a\} \\ \llbracket e_1 \cup e_2 \rrbracket &\stackrel{\text{def}}{=} \llbracket e_1 \rrbracket \cup \llbracket e_2 \rrbracket \\ \llbracket e_1 \cdot e_2 \rrbracket &\stackrel{\text{def}}{=} \llbracket e_1 \rrbracket \cdot \llbracket e_2 \rrbracket \\ \llbracket e^* \rrbracket &\stackrel{\text{def}}{=} \llbracket e \rrbracket^* \end{aligned}$$

NB : pour  $A, B \in \mathcal{P}(V^*)$ ,  $A.B \stackrel{\text{def}}{=} \{u.v \mid u \in A \wedge v \in B\}$   
 $A^* \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} A^n$  avec  $A^0 \stackrel{\text{def}}{=} \{\epsilon\}$  et  $A^{n+1} \stackrel{\text{def}}{=} A.A^n$