Traitement informatique des langages informatiques

Sylvain.Boulme@univ-grenoble-alpes.fr

Diaporama & notes du cours TLC période 2

Ensimag – Apprentissage 1A – 2019-2020

```
git clone http://www-verimag.imag.fr/~boulme/TLC1APP/.git
```

Chapitre 1 Introduction

1.1 Contexte du cours

1.2 Introduction aux concepts de base

“Traitement informatique des langages informatiques” ?

NB : langage informatique = conçu pour rendre possible certains traitements informatiques (par opposition aux langues naturelles).

Objectifs pédagogiques du cours

Introduction aux théories de la syntaxe
et de la sémantique des langages informatiques.

En particulier

Théorie de la syntaxe et de la sémantique
des langages algébriques (hors-contextes)
& application à la conception d’interpréteurs.

Travailler en parallèle CTD (théorie) et TP (pratique)

- le TP commence directement par les applications du chapitre 3 (donc très en avance sur le cours).
- Le sujet du TP reformule (presque) entièrement le cours en articulant les différentes notions sur un cas concret.
- le CTD fait dès le chapitre 2 référence aux exemples du TP.
- En début de période 3, le CTD aide à réaliser le TP.

Liens avec les autres cours

- Automates finis (langages réguliers, automates).
- Projet GL (compilation d'un mini-Java).
- Logique (logique propositionnelle et notion de termes).
- Algorithmiques (notamment algos sur les arbres).
- Logiciel de base (trad. "haut niveau" vers "bas niveau").
- en 2A : Sémantique & analyse de programme.
- en 2A : ACVL (patrons de conception).

Ouvrages conseillés pour appronfondir le cours

Compilers: Principles, Techniques and Tools
de Aho, Lam, Sethi & Ullman (1988/2007)

Introduction to automata theory, languages, and computation
de Hopcroft, Motwani & Ullman (1979/2007)

The Definitive ANTLR4 Reference
de Terence Parr (2013)

Theories of programming languages
de John C. Reynolds (1998)

NB: livres tous disponibles dans BUs de Grenoble.
cf. http://rugbis.grenet.fr

Chapitre 1 Introduction

1.1 Contexte du cours

1.2 Introduction aux concepts de base

Notion d'interpréteur

Définition Interpréteur = programme qui prend en entrée un texte (dans un certain "*langage source*") et effectue un certain traitement spécifié par ce texte.

Exemples

- machine virtuelle java (interpréteur de fichiers "`.class`")
- compilateur java (traduit du "`.java`" en "`.class`")
- interpréteurs bash, perl, python, etc.
- programme "grep" (interpréteur d'expressions régulières).
- pdflatex (traduit du latex vers pdf).
- visualisateurs "pdf" (evince, acrobat-reader).
- navigateur web (interprète html+javascript).
- ...

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.
  démo au chapitre 4.
- ANTLR "ANother Tool for Language Recognition"
  produit Java ou C#, utilisé en ProjetGL.

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 *syntaxe 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 *une seule grosse phrase*.
Interactif	découpée en *suite de phrases* traitées en séquence.

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)*

   `\(a|d\)*b`

Sémantique = sens = signification

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

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

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

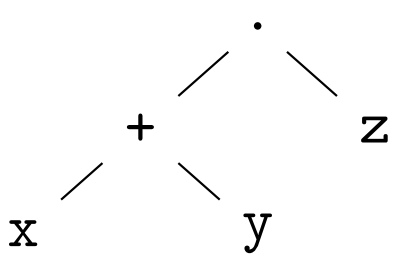
$$X = \{b\} \cup \{a, d\}.X$$

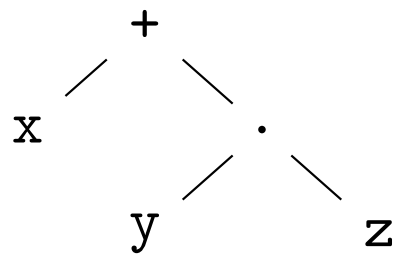
Syntaxe textuelle versus syntaxe abstraite

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 parenthèsage.
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

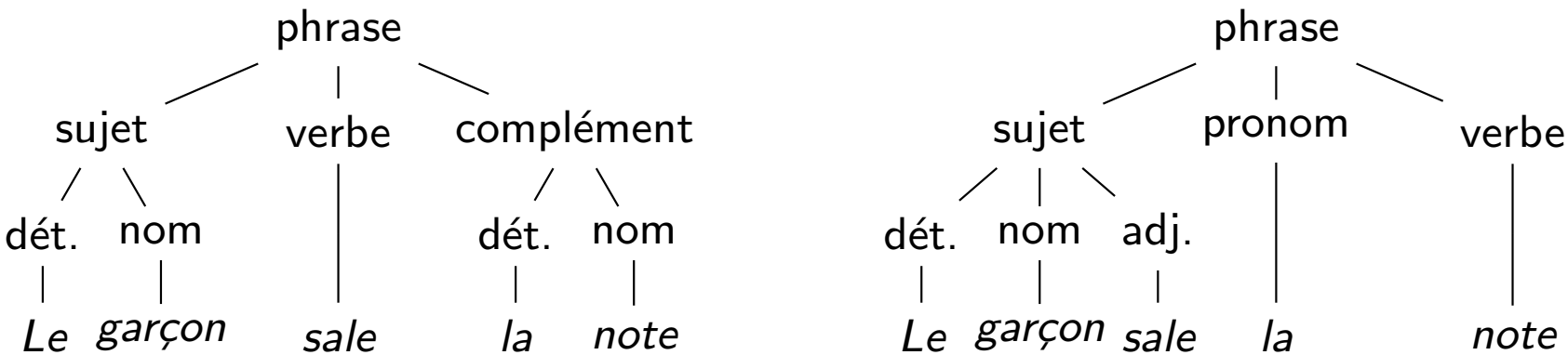
Analyse syntaxique indépendante de la sémantique !

Objectifs : traitements automatiques & modulaires & sans ambiguïtés.

Langues humaines

Interactions entre analyse syntaxique & analyse sémantique

Exemple : "*Le garçon sale la note*"



$\Rightarrow$ traitement automatique imparfait des langues naturelles !

cf. correcteurs orthographiques, traducteurs automatiques, etc.

Sémantique informelle versus formelle

Sémantique informelle
Sens des constructions du langage expliqué en langue naturelle (typiquement anglais) : manuel de référence ou norme iso.

Sémantique formelle
Fonction mathématique définie par cas sur structure d'AST.
$\Rightarrow$ "proche" du *backend* d'un interpréteur.

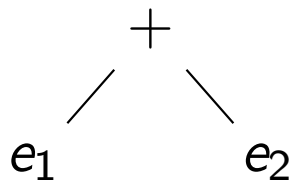
Réciproquement
Backend d'un interpréteur définit aussi une sémantique formelle mais glt trop complexe pour comprendre/raisonner !

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

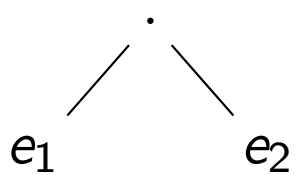
Soit V ensemble fini.

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

- constantes ϵ 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 $[\![e]\!] \in \mathcal{P}(V^*)$ (= langage reconnu par e) est définie par cas sur chaque type de nœuds :

$$[\![\emptyset]\!] \stackrel{def}{=} \emptyset$$

$$[\![\epsilon]\!] \stackrel{def}{=} \{\epsilon\}$$

$$[\![a]\!] \stackrel{def}{=} \{a\}$$

$$\left[\!\!\left[\begin{array}{ccc} & + & \\ \diagup & & \diagdown \\ e_1 & & e_2 \end{array}\right]\!\!\right] \stackrel{def}{=} [\![e_1]\!] \cup [\![e_2]\!]$$

$$\left[\!\!\left[\begin{array}{ccc} & \cdot & \\ \diagup & & \diagdown \\ e_1 & & e_2 \end{array}\right]\!\!\right] \stackrel{def}{=} [\![e_1]\!] \,.\, [\![e_2]\!]$$

$$\left[\!\!\left[\begin{array}{c} * \\ | \\ e \end{array}\right]\!\!\right] \stackrel{def}{=} [\![e]\!]^*$$

NB : pour $A, B \in \mathcal{P}(V^*)$, $A.B \stackrel{def}{=} \{u.v \mid u \in A \wedge v \in B\}$

$A^* \stackrel{def}{=} \bigcup_{n\in\mathbb{N}} A^n$ avec $A^0 \stackrel{def}{=} \{\epsilon\}$ et $A^{n+1} \stackrel{def}{=} A.A^n$