

Construction d'analyseurs syntaxiques
TL2 Ensimag 1A 2022-2023

Chapitre 1
Interprétation des langages formels

Xavier.Nicollin@grenoble-inp.fr
thanks Sylvain Boulmé et Lionel Rieg

Notion d'interpréteur (ou interprète) en informatique

Définition Interpréteur = programme qui prend en entrée un texte (dans un certain “*langage source*”) et effectue un certain traitement spécifié par ce texte conformément à la *sémantique* du langage

Exemples

- ▶ interprètes bash, perl, python...
- ▶ compilateur gcc (traduit du “.c” en exécutable)
- ▶ interprète SQL (requête sur bases de données)
- ▶ programme “grep” (interprète d’expressions régulières)
- ▶ navigateur web (interprète html+javascript)
- ▶ pdflatex (traduit du latex vers pdf)
- ▶ visualisateurs “pdf” (evince, document-viewer, acrobat-reader)
- ▶ ...

Méta-interpréteurs

Déf Méta-interpréteur = pgm pour construire des interpréteurs

Entrée : spécification *haut-niveau* d'un interpréteur

Sortie : un programme qui réalise cet interpréteur

Exemples

- ▶ YACC “Yet Another Compiler Compiler”
(outil historique fondateur, depuis les années 70)
- ▶ Bison (raffinement de Yacc)
outil GNU qui produit du C, C++ ou Java
longtemps utilisé dans implémentation de gcc
- ▶ ANTLR “ANother Tool for Language Recognition”
produit Java ou C#, utilisé en ProjetGL en 2A-Ensimag

Vers la spécification mathématique d'interpréteurs

À un niveau abstrait,

interpréteur \simeq traducteur

\simeq fonction *partielle* $\mathcal{I} : V^* \rightarrow D$

avec

$\text{dom}(\mathcal{I}) =$ syntaxe du langage d'entrée

$D =$ domaine sémantique

ens. des comportements possibles de l'interpréteur

ens. des valeurs sémantiques des éléments du langage

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)

$\backslash(a|d\backslash)^*b$

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

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

(cf. lemme d'Arden)

Chapitre 1 Interprétation des langages formels

Interpréteur simple = traducteur dirigé par la syntaxe

Structure des interpréteurs complexes et analyse syntaxique

Traduction dirigée par la syntaxe

Principe

- ▶ syntaxe donnée par une BNF (= grammaire hors-contexte)
- ▶ sémantique définie récursivement
sur la structure des arbres d'analyse (= arbres de dérivation)

Exemples

- ▶ suite du chapitre 1 :
formalisation des expressions régulières sur $\{a, b\}$
- ▶ chapitre 2 + TP :
une calculette (avec calculs sur expressions arithmétiques)
- ▶ plus généralement : *analyseurs syntaxiques*

Exemple : syntaxe des expressions régulières sur $\{a, b\}$

Syntaxe donnée par BNF (= grammaire hors-contexte)

```
er ::= 0      // vide (évite confusion avec méta)
    | 1      // epsilon (idem)
    | a
    | b
    | er + er
    | er . er
    | er *
    | ( er )
```

BNF sur vocabulaire terminal $\{0, 1, a, b, +, ., *, (,)\}$

Sémantique notée $\llbracket e \rrbracket$ pour e *arbre d'analyse*
d'un mot de racine er

définie *récursivement* sur structure de e (cf. slide suivant)

Ici on veut obtenir (sémantique) le langage représenté par l'e.r.

Sémantique des expressions régulières sur $\{a, b\}$

Déf récursive de $\llbracket e \rrbracket \in \mathcal{P}(\{a, b\}^*)$ pour e arbre d'analyse

$$\left[\begin{array}{c} \text{er} \\ | \\ 0 \end{array} \right] \stackrel{\text{def}}{=} \emptyset$$

$$\left[\begin{array}{c} \text{er} \\ | \\ a \end{array} \right] \stackrel{\text{def}}{=} \{a\}$$

$$\left[\begin{array}{c} \text{er} \\ | \\ 1 \end{array} \right] \stackrel{\text{def}}{=} \{\varepsilon\}$$

$$\left[\begin{array}{c} \text{er} \\ | \\ b \end{array} \right] \stackrel{\text{def}}{=} \{b\}$$

$$\left[\begin{array}{ccc} & \text{er} & \\ & / \quad | \quad \backslash & \\ e_1 & + & e_2 \end{array} \right] \stackrel{\text{def}}{=} \llbracket e_1 \rrbracket \cup \llbracket e_2 \rrbracket$$

$$\left[\begin{array}{ccc} & \text{er} & \\ & / \quad | \quad \backslash & \\ e_1 & \cdot & e_2 \end{array} \right] \stackrel{\text{def}}{=} \llbracket e_1 \rrbracket \cdot \llbracket e_2 \rrbracket$$

$$\left[\begin{array}{ccc} & \text{er} & \\ & / \quad \backslash & \\ e_0 & * & \end{array} \right] \stackrel{\text{def}}{=} \llbracket e_0 \rrbracket^*$$

$$\left[\begin{array}{ccc} & \text{er} & \\ & / \quad | \quad \backslash & \\ (& e &) \end{array} \right] \stackrel{\text{def}}{=} \llbracket e \rrbracket$$

BNF attribuée = syntaxe + sémantique

Profil d'attribut(s) $er \uparrow \mathcal{P}(\{a, b\}^*)$

definit $\llbracket e \rrbracket$ pour e arbre d'analyse de racine er

| | | | |
|-----------------|-------|-----------------------------------------|------------------------|
| $er \uparrow L$ | $::=$ | 0 | $L := \emptyset$ |
| | | 1 | $L := \{\varepsilon\}$ |
| | | a | $L := \{a\}$ |
| | | b | $L := \{b\}$ |
| | | $er \uparrow L_1 + er \uparrow L_2$ | $L := L_1 \cup L_2$ |
| | | $er \uparrow L_1 \cdot er \uparrow L_2$ | $L := L_1 \cdot L_2$ |
| | | $er \uparrow L_1 *$ | $L := L_1^*$ |
| | | $(er \uparrow L)$ | |

Note : dernière ligne raccourci pour

$$er \uparrow L ::= (er \uparrow L_1) \quad L := L_1$$

Exo 1 Le mot “a + b * . a” a plusieurs arbres d'analyse.
Lesquels ? Quelles sont les sémantiques associées ?

Problème BNF ambiguë \Rightarrow sémantique non-déterministe !

Priorité des opérateurs (*precedence* en anglais)

Sur expressions arithmétiques : “ $20 - 2 \times 3$ ” pourrait *a priori* représenter “ $(20 - 2) \times 3$ ” ou “ $20 - (2 \times 3)$ ”

Par convention

$$x - y \times z \stackrel{\text{def}}{=} x - (y \times z) \quad \text{et} \quad x \times y - z \stackrel{\text{def}}{=} (x \times y) - z$$

Formellement “ \times ” *plus prioritaire* que “ $-$ ”

Problème des opérateurs non associatifs :

$$(5 - 3) - 2 \neq 5 - (3 - 2) \quad \text{et} \quad (2^3)^2 \neq 2^{(3^2)}$$

Associativité à gauche $x - y - z \stackrel{\text{def}}{=} (x - y) - z$

le cas de tous opérateurs arithmétiques sauf exponentiation
(y compris les opérateurs “associatifs” $+$, \times ...)

Associativité à droite $x^{yz} \stackrel{\text{def}}{=} x^{(yz)}$

ATTENTION, associativité en fait définie par niveau de priorité

$$x + y - z = (x + y) - z \quad \text{et} \quad x - y + z = (x - y) + z$$

Sémantique déterministe via BNF attribuée + priorités

| | |
|-------------------------------------------|------------------------|
| $er \uparrow L ::= 0$ | $L := \emptyset$ |
| $ 1$ | $L := \{\varepsilon\}$ |
| $ a$ | $L := \{a\}$ |
| $ b$ | $L := \{b\}$ |
| $ er \uparrow L_1 + er \uparrow L_2$ | $L := L_1 \cup L_2$ |
| $ er \uparrow L_1 \cdot er \uparrow L_2$ | $L := L_1 \cdot L_2$ |
| $ er \uparrow L_1 *$ | $L := L_1^*$ |
| $ (er \uparrow L)$ | |

Table de priorités de er

| | | | |
|---------------------------------|---------------------|---|---------|
| niveau 2 (priorité min) | associatif à gauche | + | binaire |
| niveau 1 | associatif à gauche | . | binaire |
| niveau 0 (priorité max) | | * | unaire |

Exo 2 Donner l'arbre d'analyse de "a + b * . a" qui respecte les priorités et donner la sémantique associée

Cas général des attributs dans BNFs attribuées

Deux catégories d'attributs

- ▶ attribut \uparrow dit “synthétisé”, i.e. propagé du fils vers le père
 \Rightarrow correspond à un résultat du calcul
- ▶ attribut \downarrow dit “hérité”, i.e. propagé du père vers le fils
 \Rightarrow correspond à un paramètre du calcul

Plus formellement :

Chaque non-terminal X a un **profil d'attributs** fixe

$$X \downarrow H_1 \dots \downarrow H_n \uparrow S_1 \dots \uparrow S_m$$

qui représente une *fonction sémantique* $\llbracket t, h_1, \dots, h_n \rrbracket$

de $(T \times H_1 \times \dots \times H_n)$ vers $(S_1 \times \dots \times S_m)$

où T = ensemble des arbres d'analyse de racine X

Exemple : compiler er en automates finis (1/2)

Spécification par BNF attribuée (diapo suivante) de profil

$$\text{er} \downarrow \mathbb{N} \downarrow \mathbb{N} \uparrow \mathbb{N} \uparrow \mathcal{P}(\mathbb{N} \times \{a, b, \varepsilon\} \times \mathbb{N})$$

telle que pour tout arbre dérivant $e \downarrow i \downarrow n \uparrow f \uparrow \delta$, avec $i < n$ alors

- ▶ $f \geq n$ (donc $f \neq i$)
- ▶ l'automate $\langle \{i\} \cup [n..f], \{a, b\}, \delta, \{i\}, \{f\} \rangle$ reconnaît le langage défini par l'expression régulière e
- ▶ i (resp. f) n'a pas de transition entrante (resp. sortante)

NB f pas forcément accessible depuis i (e.g. si $e = 0$)

Construction de l'automate paramétrée par i et n avec $i < n$

Idée i : état initial pour e n : plus petit état $\neq i$ pour e
 f : état final pour e δ : relation de transition pour e

Utilise opération notée " $\delta[q \rightsquigarrow q']$ " renommant q en q' dans δ

$$\{(1, a, 2), (3, b, 1), (4, a, 2)\}[1 \rightsquigarrow 2] = \{(2, a, 2), (3, b, 2), (4, a, 2)\}$$

Compiler en automates finis (2/2)

$$\text{er} \downarrow i \downarrow n \uparrow f \uparrow \delta ::= 0$$

$$f := n; \delta := \emptyset$$

$$| \quad 1$$

$$f := n; \delta := \{(i, \varepsilon, f)\}$$

$$| \quad a$$

$$f := n; \delta := \{(i, a, f)\}$$

$$| \quad b$$

$$f := n; \delta := \{(i, b, f)\}$$

$$| \quad \text{er} \downarrow i \downarrow n \uparrow f_1 \uparrow \delta_1 \quad . \quad \text{er} \downarrow f_1 \downarrow (f_1 + 1) \uparrow f \uparrow \delta_2$$

$$\delta := \delta_1 \cup \delta_2$$

$$| \quad \text{er} \downarrow i \downarrow n \uparrow f_1 \uparrow \delta_1 \quad + \quad \text{er} \downarrow i \downarrow f_1 \uparrow f \uparrow \delta_2$$

$$\delta := \delta_1[f_1 \rightsquigarrow f] \cup \delta_2$$

$$| \quad \text{er} \downarrow i \downarrow n \uparrow f_1 \uparrow \delta_1 \quad *$$

$$f := f_1 + 1; \delta := \delta_1[i \rightsquigarrow f_1] \cup \{(i, \varepsilon, f_1), (f_1, \varepsilon, f)\}$$

$$| \quad (\text{er} \downarrow i \downarrow n \uparrow f \uparrow \delta)$$

Exercice

Exo 3 Donner l'arbre d'analyse de l'e.r. "a . b . b + a *"
qui respecte les priorités, puis

- ▶ Propager les attributs quand $i = 2$ et $n = 5$ à la racine de l'arbre
- ▶ Dessiner l'automate obtenu

Quel automate aurait-on obtenu en prenant $i = 0$ et $n = 1$ à la racine de l'arbre ?

Remarques

- ▶ il n'est pas nécessaire ici de comprendre pourquoi la BNF produit un automate correct
- ▶ il faut juste exécuter cette BNF attribuée sur l'exemple
- ▶ pour d'autres exos sur cette BNF : cf. exam de mai 2018

Chapitre 1 Interprétation des langages formels

Interpréteur simple = traducteur dirigé par la syntaxe

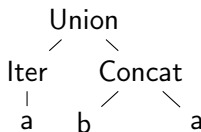
Structure des interpréteurs complexes et analyse syntaxique

Sémantiques complexes

► Arbres abstraits

$a * + (b . a)$
 $(a *) + (b . a)$
 $(a * + b . a)$

arbres d'analyse
différents
mais même
« structure »



AST (Abstract Syntax Tree) : abstraction des arbres d'analyse

- AST construit depuis *texte d'entrée* par *analyseur syntaxique* (*parser* en anglais)
- Sémantiques complexes \Rightarrow
 enchaînement de traitements sur AST
 cf. projetGL en 2A-Ensimag

Introduction à l'analyse syntaxique

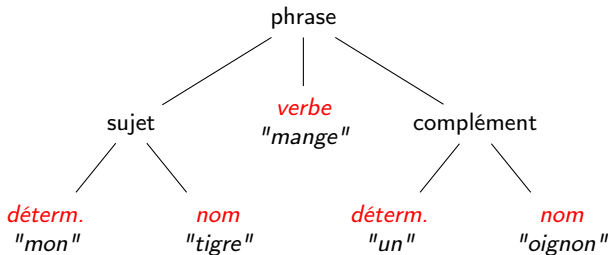
Analogie avec le français pour lire

"Mon_tigre_mange__un_oignon"

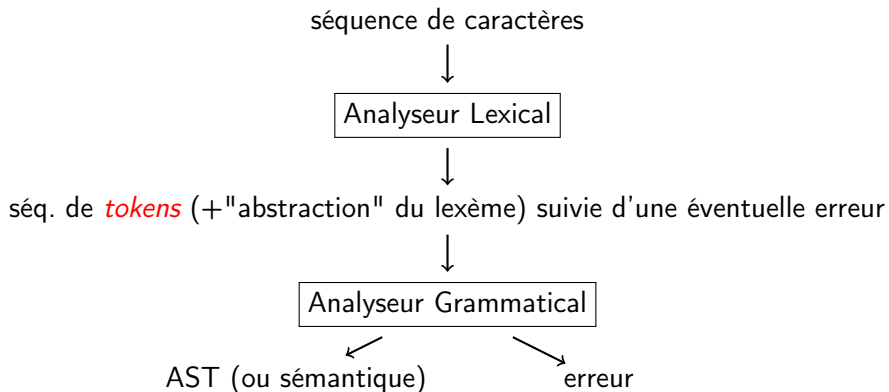
- 1) Analyse lexicale découpe en *mots* ("*lexèmes*, *unités lexicales*")
et détermine leur *nature* ("*token*")

| | | | | |
|----------------|------------|--------------|----------------|------------|
| <i>déterm.</i> | <i>nom</i> | <i>verbe</i> | <i>déterm.</i> | <i>nom</i> |
| Mon | _tigre | _mange | _ _un | _oignon |

- 2) Analyse grammaticale détermine la *structure* de la *phrase*



Architecture d'un analyseur syntaxique

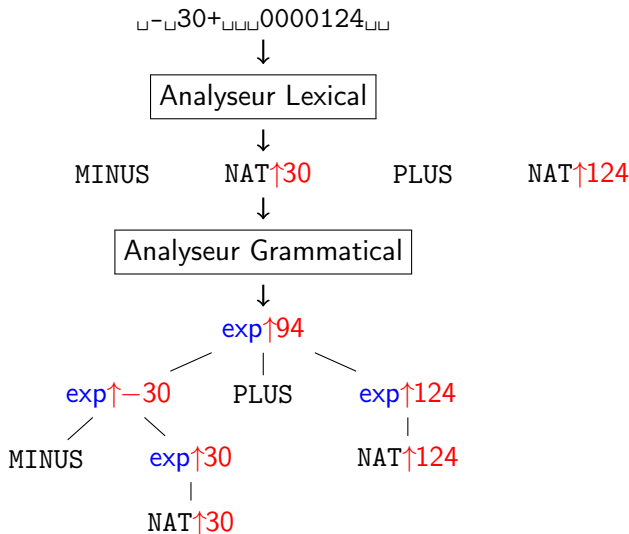


lexème (unité lexicale) = un mot formé d'une suite de caractères

Analyse grammaticale = traduction dirigée par syntaxe via BNF
dont chaque terminal (***token***) dénote un *ensemble de lexèmes*

Exemple sur expressions arithmétiques du chapitre 2

Analyse de la ligne ci-dessous où espaces matérialisés par \square



Principes de l'analyse lexicale

- ▶ Lecture du prochain lexème à la **demande** de l'analyseur grammatical (origine du mot “*token*”, jeton...)
Dispense de stocker toute la séquence des lexèmes en mémoire
- ▶ Langage d'entrée inclus dans “ $(\text{SEP} \cup \text{TOKEN}_1 \cup \dots \cup \text{TOKEN}_n)^*$ ”
où SEP , TOKEN_1 , ..., TOKEN_n sont des langages 2 à 2 disjoints
 - ▶ SEP : un langage *régulier* de “séparateurs”
 - ▶ TOKEN_1 , ..., TOKEN_n : des langages *disjoints réguliers* pour chaque lexème (\leadsto terminaux de la grammaire)
- ▶ *Lecture “gauche/droite” des lexèmes*
prochain lexème =
+ **long préfixe** $\in \text{SEP}^* . (\text{TOKEN}_1 \cup \dots \cup \text{TOKEN}_n)$
Exemple sur “`00124+1`”, lire “`00124`” comme un lexème
et renvoyer NAT (le token)
en ayant calculé l'entier 124 (la *valeur*) à la volée...

\Rightarrow analyseur lexical = automate fini

En conclusion

- ▶ Interpréteur généralement composé d'un *analyseur syntaxique* qui produit un AST et d'un "*back-end*" qui effectue les traitements à partir de l'AST
- ▶ L'*analyseur syntaxique* inclut un *analyseur lexical* (analyseur syntaxique *identifié avec* analyseur grammatical)
- ▶ *Construction automatisée* des analyseurs lexicaux et syntaxiques à partir de spécifications haut-niveau (expressions régulières et BNF attribuées)

Cadre des premières semaines de TL2

Étude de *l'analyse syntaxique*

NB “*automatisée*” \neq “*automatique*”

Exemple indécidabilité de l'ambiguïté des BNF

Nécessité de comprendre comment fonctionnent les outils
pour mieux “gérer” leur limites

En TP : programmation “manuelle” d'un analyseur syntaxique

Utilisation de ANTLR en ProjetGL de 2A-Ensimag