Construction d'analyseurs syntaxiques
(*An introduction to parsing*)
TL2 Ensimag 1A 2020-2021

Chapitre 1
Interprétation des langages formels

Xavier.Nicollin@grenoble-inp.fr
thanks Sylvain Boulmé

Notion d'interpréteur 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" (interpréte d'expressions régulières)
- navigateur web (interprète html+javascript)
- pdflatex (traduit du latex vers pdf)
- visualisateurs "pdf" (evince, 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* $\mathcal{I} : V^* \to D$

avec
$\mathrm{dom}(\mathcal{I})$ = 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)*
   `\(a|d\)*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

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

(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 $\{\mathtt{a}, \mathtt{b}\}$
- chapitre 2 + TP :
  une calculette (avec calculs sur expressions arithmétiques)
- plus généralement : *analyseurs syntaxiques*

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

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 $\{\mathtt{0}, \mathtt{1}, \mathtt{a}, \mathtt{b}, \mathtt{+}, \mathtt{.}, \mathtt{*}, \mathtt{(}, \mathtt{)}\}$

Sémantique notée $[\![e]\!]$ 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 $\{\mathtt{a}, \mathtt{b}\}$

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

$$\left[\!\!\left[\begin{array}{c}\mathtt{er}\\|\\\mathtt{0}\end{array}\right]\!\!\right] \stackrel{def}{=} \emptyset \qquad \left[\!\!\left[\begin{array}{c}\mathtt{er}\\|\\\mathtt{a}\end{array}\right]\!\!\right] \stackrel{def}{=} \{a\}$$

$$\left[\!\!\left[\begin{array}{c}\mathtt{er}\\|\\\mathtt{1}\end{array}\right]\!\!\right] \stackrel{def}{=} \{\varepsilon\} \qquad \left[\!\!\left[\begin{array}{c}\mathtt{er}\\|\\\mathtt{b}\end{array}\right]\!\!\right] \stackrel{def}{=} \{b\}$$

$$\left[\!\!\left[\begin{array}{ccc}&\mathtt{er}&\\ \swarrow&|&\searrow\\ e_1&+&e_2\end{array}\right]\!\!\right] \stackrel{def}{=} [\![e_1]\!] \cup [\![e_2]\!]$$

$$\left[\!\!\left[\begin{array}{ccc}&\mathtt{er}&\\ \swarrow&|&\searrow\\ e_1&\cdot&e_2\end{array}\right]\!\!\right] \stackrel{def}{=} [\![e_1]\!] . [\![e_2]\!]$$

$$\left[\!\!\left[\begin{array}{ccc}&\mathtt{er}&\\ \swarrow&&\searrow\\ e_0&&*\end{array}\right]\!\!\right] \stackrel{def}{=} [\![e_0]\!]^* \qquad \left[\!\!\left[\begin{array}{ccc}&\mathtt{er}&\\ \swarrow&|&\searrow\\ (&e&)\end{array}\right]\!\!\right] \stackrel{def}{=} [\![e]\!]$$

BNF attribuée = syntaxe + sémantique

Profil d'attribut(s) $\mathtt{er}\uparrow\mathcal{P}(\{a, b\}^*)$
definit $[\![e]\!]$ pour e arbre d'analyse de racine er

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

Note : dernière ligne raccourci pour
$\mathtt{er}\uparrow L$ ::= ($\mathtt{er}\uparrow L_1$) $L := L_1$

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

Problème BNF ambiguë ⇒ 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
$x - y \times z \stackrel{def}{=} x - (y \times z)$ et $x \times y - z \stackrel{def}{=} (x \times y) - z$

Formellement "$\times$" *plus prioritaire* que "$-$"

Problème des opérateurs non associatifs :
$(5 - 3) - 2 \neq 5 - (3 - 2)$ et $(2^3)^2 \neq 2^{(3^2)}$

Associativité à gauche $x - y - z \stackrel{def}{=} (x - y) - z$
le cas de tous opérateurs arithmétiques sauf exponentiation
(y compris les opérateurs "associatifs" $+$, $\times$...)

Associativité à droite $x^{y^z} \stackrel{def}{=} x^{(y^z)}$

ATTENTION, associativité en fait définie par niveau de priorité
$x + y - z = (x + y) - z$ et $x - y + z = (x - y) + z$

Sémantique déterministe via BNF attribuée + priorités

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

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
$\mathbf{X}{\downarrow}H_1 \ldots {\downarrow}H_n{\uparrow}S_1 \ldots {\uparrow}S_m$
qui représente une *fonction sémantique* $[\![t, h_1, \ldots, h_n]\!]$
de $(T \times H_1 \times \ldots \times H_n)$ vers $(S_1 \times \ldots \times S_m)$
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

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

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 **er** en automates finis (2/2)

$\mathbf{er}{\downarrow}i{\downarrow}n{\uparrow}f_1{\uparrow}\delta_1$ ::=

0
$f := n;\ \delta := \emptyset$

| 1
$f := n;\ \delta := \{(i, \varepsilon, f)\}$

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

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

| $\mathbf{er}{\downarrow}i{\downarrow}n{\uparrow}f_1{\uparrow}\delta_1$. $\mathbf{er}{\downarrow}f_1{\downarrow}(f_1{+}1){\uparrow}f{\uparrow}\delta_2$
$\delta := \delta_1 \cup \delta_2$

| $\mathbf{er}{\downarrow}i{\downarrow}n{\uparrow}f_1{\uparrow}\delta_1$ + $\mathbf{er}{\downarrow}i{\downarrow}f_1{\uparrow}f{\uparrow}\delta_2$
$\delta := \delta_1[f_1 \rightsquigarrow f] \cup \delta_2$

| $\mathbf{er}{\downarrow}i{\downarrow}n{\uparrow}f_1{\uparrow}\delta_1$ *
$f := f_1{+}1;\ \delta := \delta_1[i \rightsquigarrow f_1] \cup \{(i, \varepsilon, f_1), (f_1, \varepsilon, f)\}$

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

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

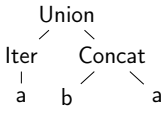
Structure des interpréteurs complexes et analyse syntaxique

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

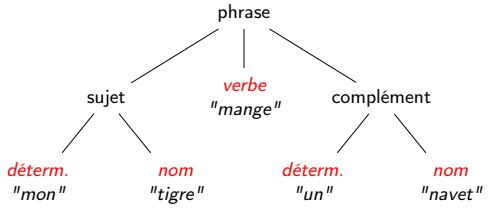
Introduction à l'analyse syntaxique

Analogie avec le français pour lire
"Mon␣tigre␣mange␣␣un␣navet"

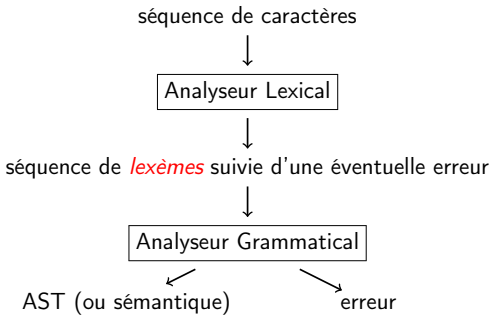
1) Analyse lexicale découpe en *mots* ("*lexèmes*, *unités lexicales*") et détermine leur *nature* ("*token*")

déterm.	*nom*	*verbe*	*déterm.*	*nom*
Mon	␣tigre	␣mange	␣␣un	␣navet

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



Architecture d'un analyseur syntaxique

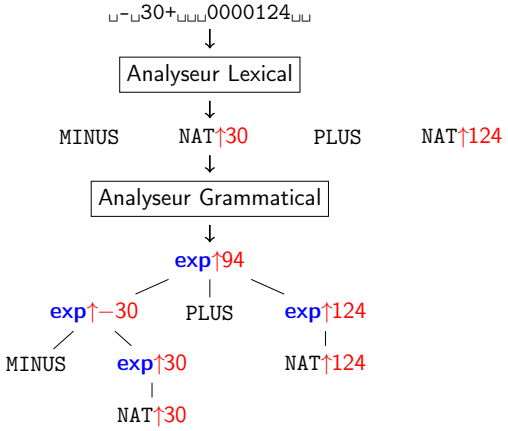


***lexème* (unité lexicale)** = un mot formé d'une suite de caractères

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 ␣



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 "$(\mathtt{SEP} \cup \mathtt{TOKEN}_1 \cup \ldots \cup \mathtt{TOKEN}_n)^*$" où SEP, $\mathtt{TOKEN}_1$, ..., $\mathtt{TOKEN}_n$ sont des langages 2 à 2 disjoints
  - SEP : un langage *régulier* de "*séparateurs*"
  - $\mathtt{TOKEN}_1$, ..., $\mathtt{TOKEN}_n$: des langages disjoints *réguliers* pour chaque lexème ($\rightsquigarrow$ terminaux de la grammaire)
- *Lecture "gauche/droite" des lexèmes*
  prochain lexème =
  + **long préfixe** $\in \mathtt{SEP}^*.(\mathtt{TOKEN}_1 \cup \ldots \cup \mathtt{TOKEN}_n)$
  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 en deux étapes...

Utilisation de ANTLR en ProjetGL de 2A-Ensimag