

Syntaxe et grammaires formelles

Didier Buchs

Université de Genève

9 mars 2015

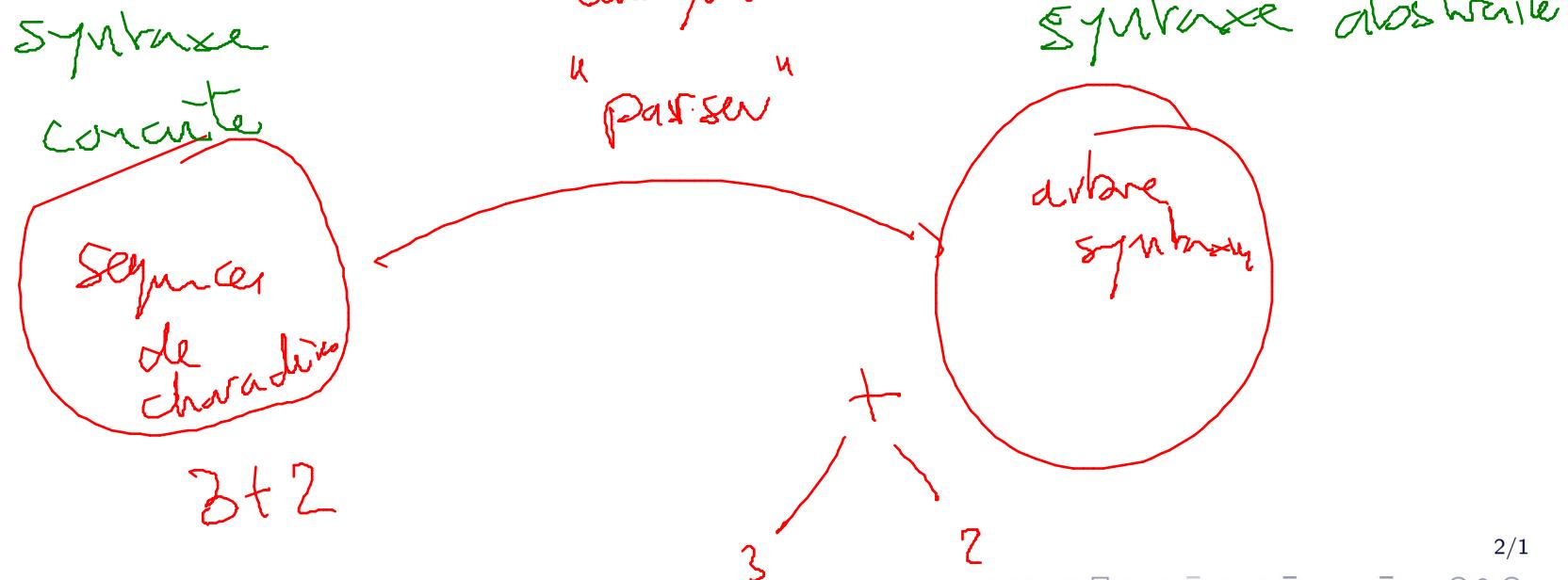
1/1



Syntaxe et grammaires formelles

- Analyse lexicale et syntaxique
 - Expressions régulières
 - Automates
- Méthodes formelles pour décrire la syntaxe
 - Classification de Chomsky
 - EBNF

Extended
Backus-Naur
form



Langage et ambiguïtés

- Le langage est un ensemble de conventions dont la signification doit être comprise par les deux parties (le lecteur et le rédacteur).
- Il faut éviter la possibilité d'affecter plusieurs significations à un même message (les ambiguïtés)
 - *j'ai vu (le chasseur) avec des jumelles*
- C'est un objectif incontournable dans les langages informatiques ! (à cause de l'automatisation)
- Comment décrire complètement un langage ?
 - Syntaxe : grammaires, diagrammes syntaxiques
 - Sémantique : opérationnelle, axiomatique, dénotationnelle

La syntaxe

- La syntaxe est la partie visible du langage
- Les règles de syntaxe définissent la forme des phrases admises dans le langage
 - syntaxe = lexèmes + structure syntaxique
- Lexèmes : unités syntaxiques élémentaires
 - mots-clés, identificateurs, opérateurs, séparateurs,
- Structure syntaxique : spécification des séquences admissibles de lexèmes
 - notion commune de grammaire
 - peut inclure des notions de mise en page (assembleurs et langages évolués primitifs)

Analyse lexicale

But : reconnaître les lexèmes d'une phrase. Il est plus facile de décrire et de lire la description des lexèmes sous la forme d'expressions régulières (approche déclarative) que sous la forme d'un algorithme (approche opérationnelle).

Exemple :

blancs	$[\backslash t \backslash n]^+$
lettre	$[A - Za - z]$
chiffre	$[0 - 9]$
ident	$\{lettre\}(- \{lettre\} \{chiffre\})^*$
entier	$\{chiffre\}^+$

lex em c

Expressions régulières

Voici une syntaxe d'expressions régulières assez commune (syntaxe de l'outil Lex) :

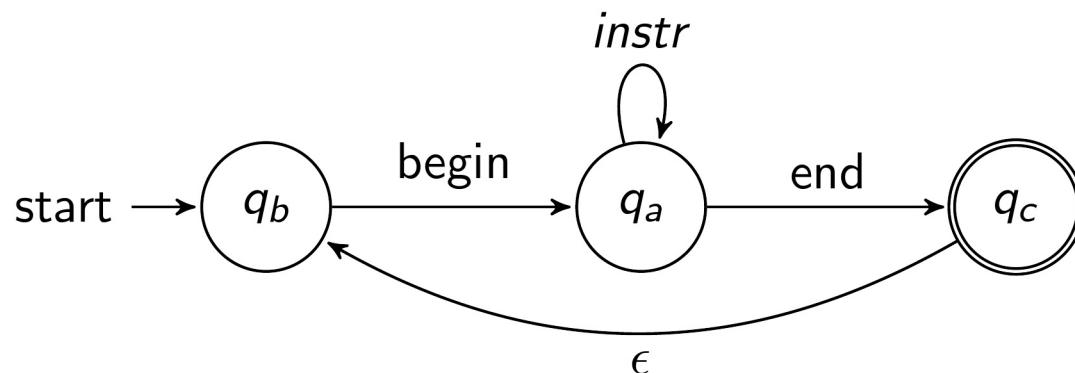
[xz]	le caractère x ou le caractère z
[x – z]	un caractère dans l'intervalle lexicographique x à z
\n \t	respectivement la fin de ligne et la tabulation
.	(le point) tout caractère sauf la fin de ligne (n)
x	le caractère x, même si c'est un symbole prédéfini
^ e	e (expression ou caractère) est en début de ligne
e\$	e est à la fin d'une ligne
e?	e est optionnel
e*	e apparaît 0 fois ou plus
e+	e apparaît 1 fois ou plus
e1 e2	e1 ou e2

Quelques automates finis correspondants

- Les cercles (éventuellement étiquetés) sont les états possibles ; les cercles concentriques sont des états finaux.
 - Les arcs (toujours étiquetés) sont les transitions entre deux états.

Exemple :

{begininstr*end}+



Expressions régulières et automates finis

- Les automates finis (ou automates à états finis) permettent de formaliser les séquences d'états et d'opérations requis pour implémenter des expressions régulières (il y a donc équivalence).
- L'outil Lex sert à produire un analyseur lexical (il génère un programme source en langage C) à partir d'expressions régulières ; l'outil Yacc en est le pendant pour l'analyse syntaxique.
- Dans la pratique, l'analyse lexicale est compliquée par divers facteurs, tels que : taille maximale des constantes numériques, commentaires enjambant plusieurs lignes ou comportant des instructions pour le compilateur (pragmas), caractères spéciaux dans les chaînes de caractères.

Vers la notion de grammaire formelle(1)

- Usuellement, la spécification de la grammaire repose sur la définition d'un certain nombre de catégories syntaxiques et des relations existant entre elles.
- Au travers des catégories syntaxiques on peut reconnaître une structure dans une phrase du langage.
- Par exemple une phrase d'une langue naturelle peut avoir la structure suivante :
 - 1. PHRASE → SUJET PREDICAT
 - 2. SUJET → PHRASE-NOMINALE
 - 3. PHRASE-NOMINALE → NOM-PROPRE
 - 4. PHRASE-NOMINALE → ARTICLE NOM-COMMUN
 - 5. PREDICAT → VERBE
 - 6. PREDICAT → VERBE PHRASE-NOMINALE

ARTICLE → le | un | les }
NOM-PROPRE → jean | pierre | paul |

9/1

Vers la notion de grammaire formelle (2)

Deux classes de symboles ont été introduites ici :

- Les catégories syntaxiques (ici en majuscules) : les symboles non terminaux.
- Les mots constitutifs de la phrase lorsque le processus de génération se termine : les symboles terminaux

Le symbole non terminal utilisé pour débuter le processus de génération (dans notre exemple : PHRASE) est appelé symbole initial (ou de départ ou encore axiome).

Le processus de génération consiste dans l'application, à chaque pas, d'une règle de réécriture appelée production, jusqu'à ce qu'aucune règle ne puisse être appliquée ou que l'on ait éliminé tous les symboles non terminaux.

Analyseurs vs grammaires génératives

- Analyseur de langage : le but est de déterminer, au moyen d'un algorithme déterministe (donc se terminant toujours par une solution au bout d'un temps fini), si une phrase donnée appartient au langage

p.ex : compilateurs (partie analyse lexicale + syntaxique)
résultats : statut (succès/échec) + arbre syntaxique

cf. cours de compilation
- Générateur de langage (formel) : ensemble de règles pour générer toutes les phrases valides possibles du langage on parle de grammaires génératives souvent plus facile à comprendre pour les humains

⇒ Il y a équivalence entre ces deux approches
(vision opérationnelle contre vision déclarative !)

Définition

Une grammaire formelle est un quadruplet (N, T, R, a) où :

- N est un ensemble fini non vide de symboles dit alphabet non terminal. Les alphabets T et N sont disjoints, leur union définit l'alphabet global : $V = N \cup T$.
- T est un ensemble fini non vide de symboles dit alphabet terminal, dont les éléments sont appelés symboles terminaux et sont ici par convention en lettres minuscules.
- R est l'ensemble fini et non vide des règles grammaticales, ou productions ; chaque production est de la forme :
 - $l \rightarrow r$ où :
 - $l \in V^*$ est appelé tête ou membre gauche, et
 - $r \in V^*$ est appelé corps ou membre droit.
 - V^* est l'ensemble de toutes les séquences formées de symboles du vocabulaire V , y compris la chaîne vide dénotée ϵ .
 - La tête a contient au moins un symbole non terminal.
- $a \in N$ constitue l'axiome ou symbole de départ.

Exemple : Le langage des expressions arithmétiques

$N = \{expr, nombres, op, chiffres\}$

$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, *, -, /\}$

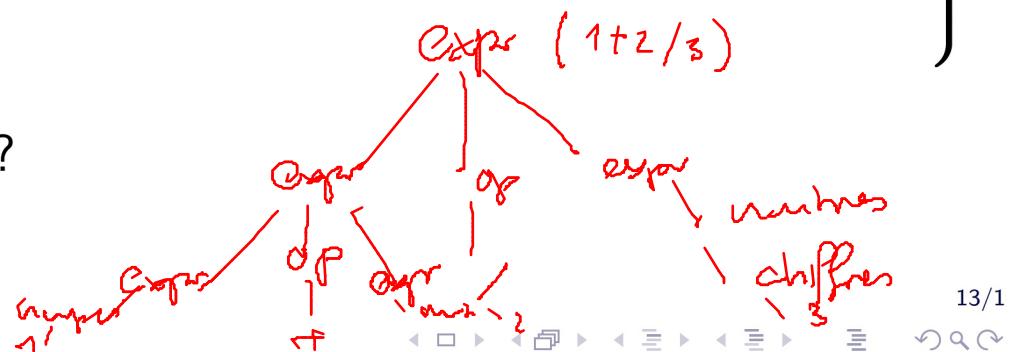
$R =$

$$\left\{ \begin{array}{lll} expr \rightarrow expr \ op \ expr & expr \rightarrow nombres & \\ op \rightarrow + & op \rightarrow - & op \rightarrow * \\ op \rightarrow / & & \\ nombres \rightarrow nombres \ chiffres & nombres \rightarrow chiffres & \\ chiffres \rightarrow 0 & chiffres \rightarrow 1 & chiffres \rightarrow 2 \\ chiffres \rightarrow 3 & chiffres \rightarrow 4 & chiffres \rightarrow 5 \\ chiffres \rightarrow 6 & chiffres \rightarrow 7 & chiffres \rightarrow 8 \\ chiffres \rightarrow 9 & & \end{array} \right\}$$

Que dire de : $1 + 2 / 3 ??$



deux autres
syntaxes



Exercice

Générer les expressions arithmétiques en Prolog :

Classification de Chomsky

- La définition des grammaires génératives donnée ci-dessus n'impose aucune contrainte sur les productions.
- En introduisant des limitations sur la forme de ces productions, Noam Chomsky a introduit en 1956 une classification hiérarchique des grammaires et des langages qui est très généralement acceptée.
- Chomsky s'intéresse avant tout aux langues naturelles, mais il n'en constitue pas moins un pionnier de l'informatique !

Classification hiérarchique de Chomsky

Les quatre types hiérarchiques de Chomsky sont :

- Le type 0 : il n'y a aucune restriction sur les productions. Pas d'algorithme d'analyse efficace correspondant
- Le type 1 : dite dépendante du contexte ; toutes les productions ont la forme : $pgq \rightarrow pdq$ où $g \in N$, $p, q \in V^*$, $d \in V^* - \{\epsilon\}$ et ϵ est la chaîne vide.
 - En d'autres termes, p et q représentent le 'contexte'. On a toujours $|a| < |b|$ (c'est-à-dire que $\text{longeur}(a) < \text{longeur}(b)$).
 - Toute grammaire dépendante du contexte possède un algorithme d'analyse syntaxique.

Classification de Chomsky (suite)

- Le type 2 : dite indépendante du contexte ; toutes les productions ont la forme

$$A \rightarrow b \text{ où } A \in N \text{ et } b \in V^*$$

- c'est-à-dire que le symbole non terminal A peut être remplacé par b indépendamment du contexte dans lequel il se trouve.
- La grande majorité des langages de programmation sont décrits par une grammaire indépendante du contexte.
- Les grammaires indépendantes du contexte ne sont pas un strict sous-ensemble des grammaires dépendantes du contexte (qui excluent la chaîne vide)

Exemple

La grammaire $G = (\{E, T, F\}, \{i, +, *, /, -, (,)\}, R, E)$ avec les règles de production $R =$

$$\left\{ \begin{array}{l} E \rightarrow T \\ E \rightarrow T + T; \quad E \rightarrow T - T \\ T \rightarrow F \\ T \rightarrow F * F; \quad T \rightarrow F/F \\ F \rightarrow i \\ F \rightarrow (E) \end{array} \right\}$$

génère le langage indépendant du contexte (noté $L(G)$) caractérisé comme contenant toutes les expressions arithmétiques de la variable i

Classification de Chomsky (suite)

- Le type 3 : dite régulière ;
 - une telle grammaire régulière peut prendre deux formes :
 - ① soit $A \rightarrow tB$ ou $A \rightarrow t$, t étant une chaîne terminale ($t \in T^*$) et A et B des symboles non terminaux. Forme appelée grammaire linéaire à droite ;
 - ② soit $A \rightarrow Bt$ ou $A \rightarrow t$, t étant une chaîne terminale ($t \in T^*$) et A et B des symboles non terminaux. Forme appelée grammaire linéaire gauche.
 - Les grammaires régulières sont un sous-ensemble des grammaires indépendantes du contexte
 - Les expressions régulières désignent la forme des règles des grammaires régulières

Prolog pour analyser et générer un langage

- Naturellement une grammaire de type 2 ou 3 peut être traduite en prolog.
 $\Rightarrow E \rightarrow E_1E_2\dots E_n$ est directement traduite en
`c :- c1, c2, ..., cn`
- Néanmoins la consommation des jetons nécessite de gérer des listes implicitement entre chaque règles. Une simplification de l'écriture est prédefinie en prolog qui permet de gérer production et consommation. La règle prolog précédente devient.
 $\Rightarrow c \dashrightarrow c_1, c_2, \dots, c_n$.
- les symboles terminaux sont des listes avec atomes [a]
- Les requêtes sont de la forme
`c([a, b, c], [c]).`
Où la liste [a, b, c] est la liste avant consommation et la liste [c] après la consommation.

20/1



Exemple

Construire un langage pour déplacer un robot, basé sur les suites d'actions *left* et *right*

Exemple (fin)

Grammaire :

`moveseq --> move.`

`moveseq --> move,moveseq.`

`move --> [left].`

`move --> [right].`

Utilisation en compilation

- Les analyseurs lexicaux (scanners) sont en principe produits sur la base d'une grammaire régulière
 - ⇒ Ce qui permet une implémentation sous forme de machine à états finis
- Les analyseurs syntaxiques (parsers) sont en principe produits à partir d'une grammaire indépendante du contexte (règles au format EBNF)
 - ⇒ Règles de même type que les expressions régulières, mais avec la récursion en plus, ce qui permet d'exprimer des structures imbriquées
 - ⇒ Le but est d'avoir une implémentation sous forme de machine à pile déterministe (= automate avec mémoire LIFO, Last In First Out)
 - ⇒ Deux types d'algorithmes existent : LL(k) (descendant) ou LR(k) (ascendant), avec prédition sur k symboles ($k \in \mathbb{N}$)