

# Grammaires et Analyse Syntaxique - Cours 1

## Introduction et Rappels

Ralf Treinen



Université  
Paris Cité



treinen@irif.fr

25 janvier 2024

## Organisation

- ▶ 12 cours d'amphi
- ▶ 12 séances de TP/TD. Première séance de TP/TD : semaine du 29/1.
- ▶ Chaque semaine soit une séance TD, soit séance TP (sur le même créneau d'horaire).
- ▶ Page web (transparents du cours, calendriers, ...)  
`http://www.irif.fr/~treinen/teaching/gas6/`
- ▶ Page moodle (pour les TP/TD) :  
`https://moodle.u-paris.fr/course/view.php?id=1643`  
Inscrivez-vous sur moodle pour votre groupe de TD!

## Contrôle de connaissances

- ▶ Examen écrit
- ▶ Un contrôle TD
- ▶ Un projet de programmation



$$session1 = 20\%TD + 30\%projet + 50\%examen1$$



$$session2 = \max(examen2, 20\%TD + 30\%projet + 50\%examen2)$$

## Pré-requis

1. *Automates et Analyse Lexicale* du L2 (il y aura des rappels de l'essentiel)
  - ▶ expressions rationnelles (comprendre le sens d'une expression rationnelle, écrire des expressions rationnelles dans des cas simples)
  - ▶ automates finis déterministes, non-déterministes et avec epsilon-transitions, élimination des epsilon-transitions, algorithme de déterminisation
  - ▶ limitation des automates et expressions rationnelles, en particulier le lemme d'itération (pumping lemma)
  - ▶ utilisation d'un générateur d'analyse lexicale du type LEX
2. *Programmation Fonctionnelle* (langage OCaml) du L3

## Qu'est-ce que c'est ce cours ?

- ▶ Qu'est-ce que c'est une *grammaire* ?
- ▶ Qu'est-ce que c'est, l'*analyse syntaxique* ?
- ▶ C'est pour analyser quoi ?
- ▶ Qu'est-ce qu'on obtient par l'analyse ?
- ▶ Quoi faire avec le résultat de cette analyse ?

## Grammaires

- ▶ Souvent utilisées pour définir des langages *structurés*.
- ▶ Les *expressions rationnelles* (voir cours AAL3) sont déjà un outil pour définir des langages, mais elles ne permettent pas d'exprimer des *structures imbriquées* intéressantes.
- ▶ Extrait simplifié d'une grammaire (du manuel OCaml) :

$$\begin{array}{l}
 \text{expr} ::= \text{constant} \\
 \quad | \quad ( \text{expr} ) \\
 \quad | \quad \text{expr operator expr}
 \end{array}$$

définit le langage des expressions *parenthésées*.

- ▶ On peut définir ce qu'est *constant* et *operator* par des expressions rationnelles.
- ▶ Nous allons utiliser une notation un peu plus “matheuse” pour les grammaires (cours 3)

## Analyse Syntaxique

- ▶ Elle va d'abord nous dire, pour une grammaire donnée, si un texte d'entrée correspond à la grammaire ou pas.
- ▶ Exemple :

$$\begin{array}{l} \text{expr} ::= \text{constant} \\ \quad | \quad ( \text{ expr } ) \\ \quad | \quad \text{expr operator expr} \end{array}$$

- ▶  $( ( 17 + 4 ) * 42 )$  est correct
- ▶  $( + 5$  n'est pas correct
- ▶ Plus important : quand le texte est correct, l'analyse syntaxique va nous indiquer la *structure* du texte !
- ▶ Il s'agit donc de la *découverte de la structure* dans un texte, selon une grammaire donnée.

## C'est pour analyser quoi ?

- ▶ Des textes, par exemple le contenu d'un fichier, qui est censé correspondre à une certaine grammaire
- ▶ Exemple (le plus important pour nous) : des programmes qui doivent correspondre à une grammaire pour OCaml, Java, ...
- ▶ Exemple : des langages de données : XML, Yaml, JSON, ...
- ▶ De toute façon il s'agit (dans ce cours) toujours de langages *artificiels* définis par des informaticiens, grâce aux grammaires.
- ▶ L'analyse des langues *naturelles* est beaucoup plus complexe, et un sujet important de la *Linguistique*.



## Qu'est-ce qu'on obtient de l'analyse ?

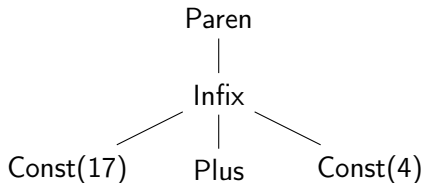
- ▶ Si l'entrée n'est pas acceptée : un message d'erreur (si possible avec une indication utile de l'erreur)
- ▶ Si l'entrée est correcte : une représentation de la structure trouvée dans le texte.
- ▶ Par exemple pour les expressions parenthésées, on peut utiliser les type OCaml suivant :

```
type constant = int
type operator = Plus | Mult
type expr = Const of constant
           | Paren of expr
           | Infix of expr * operator * expr
```

- ▶ On parle d'un *arbre de syntaxe*

## Exemple : expressions parenthésées

- ▶ Entrée : ( 17 + 4 )
- ▶ La valeur retournée par l'analyse (du type expr) :  
Paren ( Infix ( Const (17) , Plus , Const (4) ) )
- ▶ Structure d'arbre :

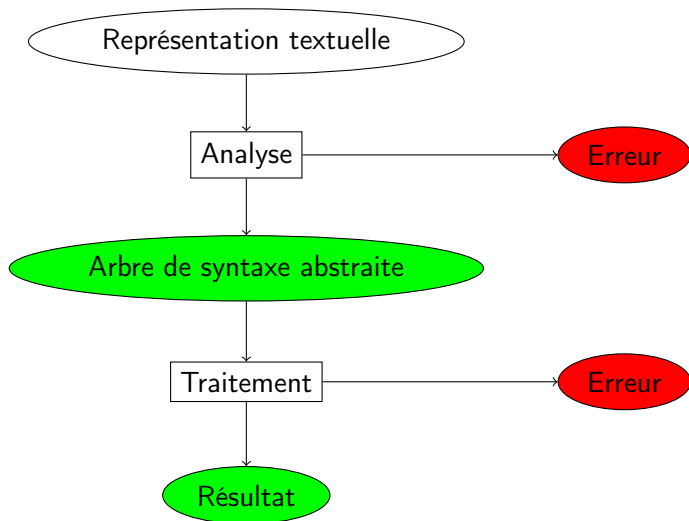


## Quoi faire avec le résultat de l'analyse ?

Ce qu'il faut faire avec l'arbre de syntaxe dépend de l'application :

- ▶ Programmes : analyser (vérifier les types, par ex.), optimiser, engendrer du code exécutable.  
C'est la *Compilation* (→ Cours du M1)
- ▶ Des données : différents traitements possibles, par exemple création d'un rendu graphique.

## Le rôle de l'analyse

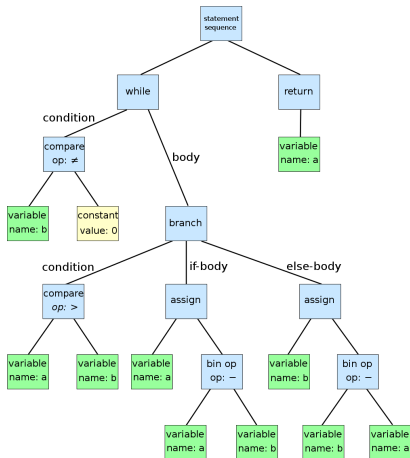


## Exemple : Un morceau d'un programme

```
while b != 0
    if a > b
        a := a - b
    else
        b := b - a
return a
```

Donné en entrée à l'analyse.

## Arbre de syntaxe (abstraite)



Résultat de l'analyse syntaxique.

## Document HTML (écrit à la main)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf8">
  <title>Programmation Fonctionnelle Avancée</title>
</head>

<body>

<center>
<h1>M1 2014/2014 : Programmation Fonctionnelle Avancée</h1>
<a href="http://www.univ-paris-diderot.fr">Université Paris-Diderot</a>
<a href="http://www.informatique.univ-paris-diderot.fr">UFR d'Informati

</center>

<h2><a name=salle>Salle et Horaires</a></h2>
Cours : Jeudi, 15h30-17h30, salle 247E, Halle aux Farines.
Premier cours: jeudi 18 septembre.
```

# Document HTML : rendu par firefox

## **M1 2014/2014 : Programmation Fonctionnelle Avancée**

[Université Paris-Diderot, UFR d'Informatique](#)

### **Salle et Horaires**

Cours : Jeudi, 15h30-17h30, salle 247E, Halle aux Farines. Premier cours: jeudi 18 septembre.

TD/TP : Mercredi, 13h30-15h30, salle 2032, bâtiment *Sophie Germain* Premier TD/TP: le mercredi 24 septembre.

### **Projet**

Voir [la page du projet](#).

### **Examen**

Jeudi, 15 janvier 2015, 12h30-15h30, amphi 5C, Halle aux Farines.

[\[Planning des examens M1 premier semestre\]](#)

### **Contenu du cours**

La programmation fonctionnelle est née presque en même temps que la programmation impérative, avec le langage Lisp à la fin des années 1950. Utilisé paradigme de programmation privilégié dans les années 1970 à 1990 pour l'Intelligence Artificielle, elle demandait des machines puissantes et chères, et



## Document XML : OpenStreetMap

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6"
  copyright="OpenStreetMap and contributors"
  license="http://opendatacommons.org/licenses/odbl/1-0/">
  <way id="62378611"
    visible="true" version="8" changeset="20691652"
    timestamp="2014-02-21T11:23:38Z" user="thibdrev" uid="1279506">
    <nd ref="779143878"/>
    <nd ref="2198721646"/>
    <nd ref="2198721727"/>
    .....
    <tag k="amenity" v="university"/>
    <tag k="building" v="yes"/>
    <tag k="name" v="Halle aux Farines (Université Paris Diderot)"/>
    <tag k="source" v="cadastre-dgi-fr source : Direction Générale des
    <tag k="wheelchair" v="yes"/>
    <tag k="wikipedia" v="fr:Université Paris VII - Diderot"/>
  </way>
</osm>
```

## Document XML : rendu par OpenStreetMap



## Ce qu'on va apprendre dans ce cours

- ▶ Comprendre et écrire des grammaires
- ▶ Plusieurs approches à l'analyse syntaxique
- ▶ Utilisation d'un générateur d'analyseurs syntaxiques
- ▶ Réaliser la première partie d'un compilateur, en utilisant :
  - ▶ un générateur d'analyse lexicale (ocamllex)
  - ▶ un générateur d'analyse grammaticale (menhir)
  - ▶ un peu de programmation en OCaml
- ▶ Les propriétés des *langages algébriques*, les *automates à pile*, et leur relation avec les langages réguliers et les automates finis.

## Démarche

- ▶ La tâche n'est pas triviale : le programme doit reconnaître la *structure* dans un texte, et générer une représentation de cette structure.
- ▶ Découpage de l'analyse en deux phases : *analyse lexicale* et *analyse grammaticale*.
- ▶ La première phase utilise les *expressions rationnelles* vues dans le cours *Automates et Analyse Lexicale*.
- ▶ La deuxième phase est le sujet principal de ce cours.

## Un autre problème : *Séquentialisation*

- ▶ Opération dans l'autre sens : traduire une représentation machine (arbre de syntaxe) en texte.
- ▶ On pourrait s'attendre à ce que les deux opérations soient l'inverse l'une de l'autre mais ce n'est pas forcément le cas.
- ▶ Exemple : un compilateur de programme peut simplement ignorer les commentaires dans le programme.
- ▶ L'opération de séquentialisation est beaucoup plus simple à mettre en œuvre que l'opération d'analyse.

## Les expressions rationnelles

- ▶ Synonymes : expressions rationnelles, expressions régulières
- ▶ Une expression rationnelle définit un ensemble de mots (aussi appelé *un langage*).
- ▶ Elles sont d'abord utilisées dans la *définition* de la syntaxe des langages informatiques (langages de programmation, langages de données).
- ▶ Exemple : les règles de Java pour l'écriture du nom d'une variable, d'une valeur entière, d'une valeur flottante, d'une chaîne de caractères.
- ▶ Il faut comprendre ces règles pour savoir écrire correctement un document, mais aussi pour savoir lire (et pour écrire un programme qui sait analyser un document).

## Définition des expressions rationnelles

- ▶ Donnée : un alphabet  $\Sigma$ , c.-à-d. un ensemble fini de symboles
- ▶ On définit d'abord la *syntaxe* des expressions rationnelles :  
Définition *inductive* de l'ensemble  $\text{Rat}$ .
- ▶ Puis on définit une *sémantique*, à l'aide d'une fonction *réursive*  $\mathcal{L}(\cdot) : \text{Rat} \rightarrow P(\Sigma^*)$ .
- ▶ La sémantique associe à chaque expression rationnelle un ensemble de mots sur l'alphabet  $\Sigma$ .

## Syntaxe des Expressions Rationnelles

Définition inductive de l'ensemble Rat, étant donné un alphabet  $\Sigma$  :

- ▶  $\emptyset \in \text{Rat}$
- ▶ Pour tout symbole  $a \in \Sigma$  :  $a \in \text{Rat}$
- ▶  $\epsilon \in \text{Rat}$ .
- ▶ Si  $r_1, r_2 \in \text{Rat}$ , alors  $r_1 r_2 \in \text{Rat}$
- ▶ Si  $r_1, r_2 \in \text{Rat}$ , alors  $r_1 \mid r_2 \in \text{Rat}$
- ▶ Si  $r \in \text{Rat}$ , alors  $(r) \in \text{Rat}$
- ▶ Si  $r \in \text{Rat}$ , alors  $r^* \in \text{Rat}$
- ▶ C'est tout.



## ATTENTION

- ▶ Nous écrivons  $|$  pour l'union, pas  $+$  comme c'était fait dans le cours *Automates et Analyse Lexicale du S3*.
- ▶ Raison : c'est la convention utilisée par presque tous les outils informatiques qui travaillent avec les expressions rationnelles, et en particulier par lex dans toutes ses variantes.

## Sémantique des Expressions Rationnelles

- ▶  $\mathcal{L}(\emptyset) = \emptyset$
- ▶  $\mathcal{L}(a) = \{a\}$  pour tout  $a \in \Sigma$
- ▶  $\mathcal{L}(\epsilon) = \{\epsilon\}$
- ▶  $\mathcal{L}(r_1 r_2) = \{w_1 w_2 \mid w_1 \in \mathcal{L}(r_1), w_2 \in \mathcal{L}(r_2)\}$
- ▶  $\mathcal{L}(r_1 \mid r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$
- ▶  $\mathcal{L}((r)) = \mathcal{L}(r)$
- ▶  $\mathcal{L}(r^*) = \{w_1 \cdots w_n \mid n \geq 0, w_i \in \mathcal{L}(r)\}$

## Exemples d'expressions rationnelles

- ▶ Nous choisissons pour l'exemple  $\Sigma = \{a, b, c, d\}$
- ▶  $\mathcal{L}(abc \mid bcd) = \{abc, bcd\}$
- ▶  $\mathcal{L}(aa(b \mid c)dd) = \{aabdd, aacdd\}$
- ▶  $\mathcal{L}((a \mid b \mid c \mid d)^*)$  : l'ensemble de tous les mots sur  $\Sigma$
- ▶  $\mathcal{L}((a \mid c)^*)$  : l'ensemble de tous les mots formés des lettres  $a$  et  $c$  seulement
- ▶  $\mathcal{L}((aa)^*)$  : l'ensemble de toutes les séquences de  $a$  de longueur paire.

## Limites des expressions rationnelles

Il y a des langages qui ne sont pas réguliers, par exemple :

- ▶ L'ensemble de tous les mots qui contiennent le même nombre de **a** que de **b**
- ▶ L'ensemble de tous les palindromes
- ▶ L'ensemble de tous les mots dont la longueur est un nombre premier
- ▶ L'ensemble des expressions arithmétiques correctement parenthésées

## Pourquoi des expressions rationnelles alors ?

- ▶ Expressivité limitée, mais . . .
- ▶ On sait faire plein de choses avec, par exemple :
  - ▶ Décider l'appartenance d'un mot au langage
  - ▶ Décider vide, universalité
  - ▶ Calculer le complément par rapport à  $\Sigma^*$
  - ▶ Calculer l'intersection
  - ▶ Traduire en un automate fini

## Sucre Syntaxique

Des extensions de syntaxe qui sont pratiques, mais qui n'apportent rien à l'expressivité.

- ▶  $r?$  : soit le mot vide, soit un mot dans  $\mathcal{L}(r)$   
Abrévation pour  $\epsilon \mid r$
- ▶  $r^+$  : une séquence *non-vide* de de mots dans  $\mathcal{L}(r)$   
Abrévation pour  $rr^*$
- ▶  $r\{n, m\}$  pour  $n, m \in \mathbb{N}$  : une séquence de  $i$  mots dans  $\mathcal{L}(r)$ ,  
où  $n \leq i \leq m$   
Abrévation pour

$$\underbrace{r \cdots r}_n \mid \cdots \mid \underbrace{r \cdots r}_m$$

## Systèmes d'expressions rationnelles

- ▶ Il est souvent utile de faire référence à une expression rationnelle déjà définie.
- ▶ Exemple :

$$r_1 = (a|b) * c$$

$$r_2 = r_1 +$$

$$r_3 = r_1 e r_2$$

- ▶ C'est simplement un raccourci :

$$r_1 = (a|b) * c$$

$$r_2 = ((a|b) * c) +$$

$$r_3 = (a|b) * ce((a|b) * c) +$$

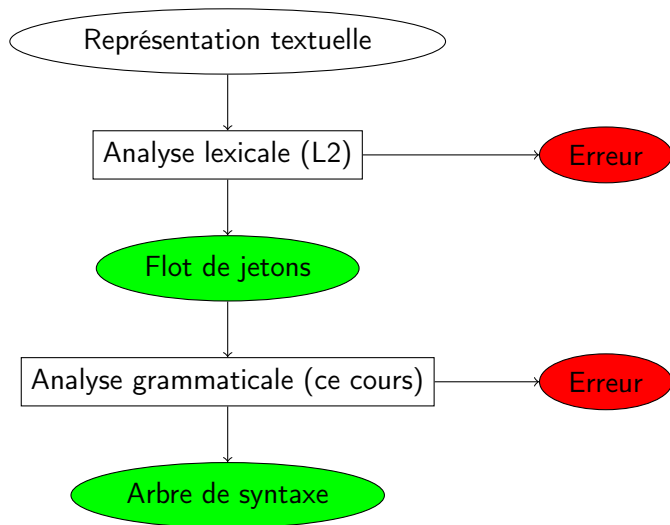
- ▶ Attention : les cycles entre définitions d'expressions rationnelles ne sont pas permis !

## L'objectif de l'analyse lexicale

- ▶ Découper un texte d'entrée en une séquence de *lexèmes*, et les représenter par des *jetons* (*tokens* en anglais)
- ▶ À la base : Classification des lexèmes qui peuvent paraître dans un texte d'entrée, à l'aide des expressions régulières.
- ▶ La phase suivante de l'analyse (l'analyse grammaticale, voir plus tard) va travailler sur le résultat de ce découpage : il s'agit d'une *abstraction* du texte d'entrée.



## Les deux phases de l'analyse

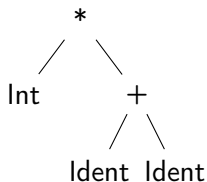


## Exemple

- ▶ Texte d'entrée :

(	7	5	6	e	2	*	(	e	5	e	7	+	v	a	l	e	u	r	2	)	)
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- ▶ Résultat de l'analyse lexicale (L2) : jetons  
PARG INT MULT PARG IDENT PLUS IDENT PARD PARD
- ▶ Résultat de l'analyse grammaticale (ce cours) : arbre



## Pourquoi deux étapes séparées ?

- ▶ On essaye de faire autant d'analyses que possible dans l'analyse lexicale.
- ▶ Raison : l'exécution d'un automate est très efficace (temps linéaire dans la longueur du texte d'entrée).
- ▶ Problème : l'expressivité des automates est limitée : théorème de l'étoile, théorème de Myhill-Nerode.
- ▶ On verra un cas concret qu'on ne peut pas reconnaître avec les expressions rationnelles la semaine prochaine.

## Jetons avec arguments

- ▶ En réalité, on veut aussi garder certaines informations avec les jetons, comme la valeur d'une constante entière, ou le nom d'un identificateur.
- ▶ Certains jetons doivent donc avoir un argument :
  - ▶ IDENT(string)
  - ▶ INT(int) (c'est bien int et pas string!)
- ▶ Séquence des jetons obtenue sur l'exemple :  
PARG INT(75600) MULT PARG IDENT("e5e7") PLUS  
IDENT("valeur2") PARD PARD

## Ignorer des informations pas pertinentes

L'analyse lexicale sert aussi à faire abstraction de certaines informations dans le texte d'entrée qui ne sont pas pertinentes pour l'analyse du texte. Souvent il s'agit de :

- ▶ Les espaces : sont utiles pour indiquer la fin d'un mot. Les espaces sont utiles *pour* l'analyse lexicale, mais une fois le découpage fait on peut les oublier.
- ▶ Les commentaires : souvent l'analyse lexicale vérifie l'écriture correcte des commentaires, mais ne les représente pas dans sa sortie.

## Exemple

Différents textes d'entrée qui peuvent donner la même séquence de jetons :

- ▶ `34 * (x + y)`
- ▶ `34*(x+y)`
- ▶ `34 *( x+ y)`
- ▶ `34 * (x+y) /* Ceci est un commentaire */`

## Quelle information retenir dans les jetons

- ▶ On retient dans les jetons seulement l'information qui est utile pour la suite.
- ▶ La distinction entre information utile/inutile dépend de l'application.
- ▶ Par exemple : Les commentaires peuvent être utiles à retenir pour certaines applications.
- ▶ Il peut être utile de conserver avec les jetons aussi des informations de *localisation* : nom du fichier source, numéro de ligne, numéro de colonne.

## Résoudre les ambiguïtés

- ▶ L'analyse lexicale va, pour produire le jeton suivant, chercher un *préfixe* du reste du texte qui correspond à une des catégories lexicales, et construire le jeton correspondant.
- ▶ Il y a deux sources d'ambiguïtés :
  - ▶ Des préfixes de longueurs différentes peuvent être reconnus
  - ▶ Les expressions régulières peuvent avoir une intersection non vide



## Préfixes de longueur différentes reconnus

Exemple :

- ▶ Catégorie lexicale :
  - ▶ IDENT : [a..z]<sup>+</sup>
- ▶ Début du texte d'entrée :  
*xyz*
- ▶ Plusieurs possibilités de découpage :
  1. IDENT("x") IDENT("y") IDENT("z")
  2. IDENT("xy") IDENT("z")
  3. IDENT("x") IDENT("yz")
  4. IDENT("xyz")
- ▶ La règle normale est : on cherche le préfixe *maximal*. Dans l'exemple ca donne IDENT("xyz").

## Plusieurs expressions régulières s'appliquent

Exemple :

- ▶ Catégories lexicales :
  - ▶ PUBLIC : `public`
  - ▶ IDENT : `[a..z]+`
  - ▶ sauter les espaces
- ▶ Début du texte d'entrée :  
`public publication`
- ▶ Plusieurs possibilités de découpage :
  1. `PUBLIC IDENT("publication")`
  2. `IDENT("public") IDENT("publication")`
- ▶ La règle normale est : à longueur égale du mot reconnu, c'est la première expression régulière qui gagne.

## Plan du cours

1. Introduction
2. Générateur d'analyse lexicale : ocamllex
3. Grammaires
4. Analyse descendante LL(1) (2 semaines)
5. Analyse ascendante LR(1) (2 semaines)
6. Générateur d'analyse grammaticale : menhir
7. Grammaires et automates à piles
8. Propriétés des langages algébriques (2 semaines)
9. Études de cas