Analyse de Données Structurées - Cours 1

Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

treinen@pps.univ-paris-diderot.fr

21 janvier 2015

Organisation

- 12 cours d'amphi
- 12 séances de TP, début la semaine du 26 janvier
- Vacances d'hiver : semaine du 23 février
- Page web du cours : `http://www.pps.univ-paris-diderot.fr/~treinen/teaching/ads4/`

Contrôle de connaissances

- Examen écrit
- Contrôle continu, consistant en
  - 20% TP noté
  - 80% projet
- $$session1 = 50\% examen1 + 50\% CC$$
- $$session2 = max(examen2, 50\% examen2 + 50\% CC)$$

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

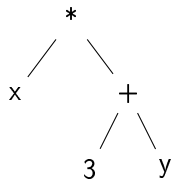
- Qu'est-ce que c'est, des *données structurées* ?
- Qu'est-ce que c'est, l'*analyse* ?
- Quoi faire avec le résultat de cette analyse ?

Données structurées

- Données qui ont une structure, souvent une structure qu'on peut imbriquer.
- Exemples : des types représentant des expressions arithmétiques, des types représentant des programmes JAVA, les arbres en général.
- En Java souvent implémentées par des objets.
- Données pas structurées : des entiers, des flottants, des chaînes de caractères (mais des valeurs de ces types peuvent faire partie de données structurées)

Exemple : expressions arithmétiques

- Expression arithmétique : $(x * (3 + y))$
- Structure d'arbre :



- Représentation en JAVA : omis pour le moment

Exemple : expressions arithmétiques

- Il y a deux niveaux différents :
  - La représentation textuelle : écrit dans un fichier, par exemple
  - La représentation par la machine : dans le cas de JAVA, des objets imbriqués (des objets qui contiennent des objets). Souvent appelé *arbre de syntaxe abstraite*.
- Modèle de la représentation machine : le dessin de l'arbre, utilisé par des humains au tableau, sur le papier.

L'analyse

- *Traduire une représentation textuelle en représentation machine*
- Traduction *automatique* par un programme
- Nous allons écrire des analyseurs en Java.
- Nous allons aussi étudier des *générateurs* qui engendrent un analyseur à partir d'une description.

Ce qu'il faut écrire

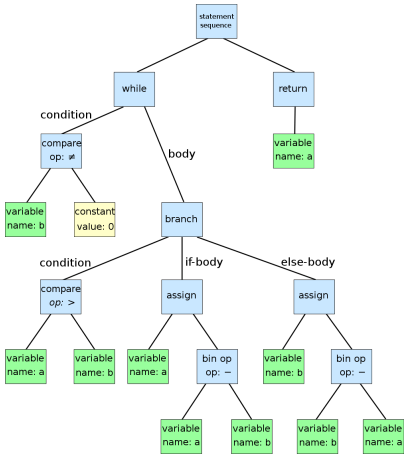
- Une définition précise des représentation textuelles qui sont légales. Pour cela on va utiliser des expressions rationnelles, des grammaires, ... (je vais vous faire des rappels de ces notions).
- Une définition des classes utilisées pour représenter les données.
- Un programme qui traduit une instance du premier (un texte) en une instance du deuxième (un objet JAVA).

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.

Séquentialisation

- Opération dans l'autre sens : traduire une représentation machine (syntaxe abstraite) en texte.
- On pourrait attendre que les deux opérations sont un l'inverse de l'autre mais ce n'est pas forcement 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.

Quoi faire avec

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

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

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'Informatique</a>

</center>

<h2><a name=salle>Salle et Horaires</a></h2>
Cours : Jeudi, 15h30-17h30, salle 247E, Halle aux Farines.
Premier cours: jeudi 18 septembre.
<p>
TD/TP : Mercredi, 13h30-15h30, salle 2032, bâtiment <i>Sophie Germain</i>
```


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]

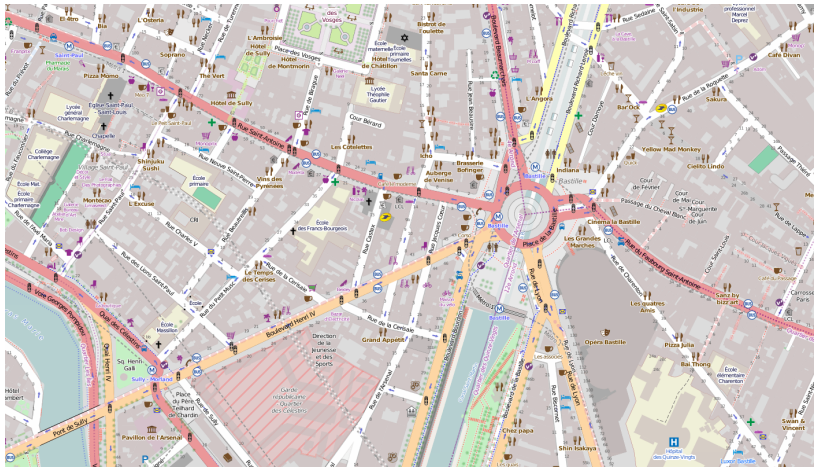
Contenu du cours

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 Impôts
    <tag k="wheelchair" v="yes"/>
    <tag k="wikipedia" v="fr:Université Paris VII - Diderot"/>
  </way>
</osm>
```

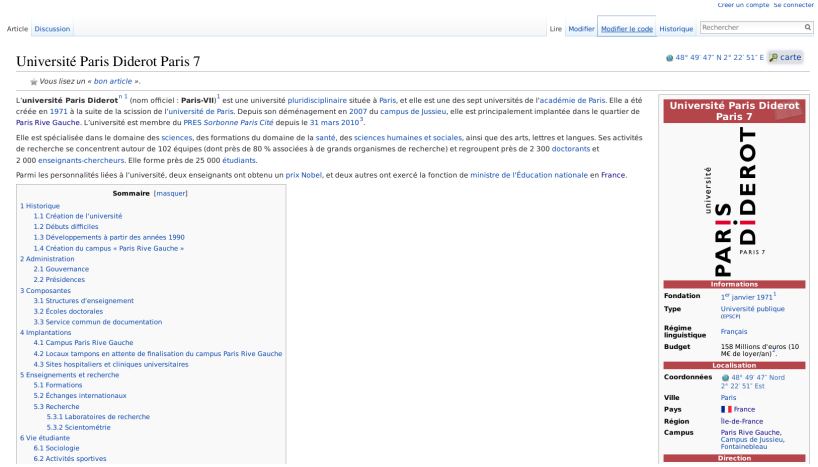
OpenStreetMap : Rendu français



Document Wiki : Wikipédia

```
{{Entête label|BA}}
{{Infobox Université
 | blason = Logo-P7.svg
 | taille blason = 120
 | nom=Université Paris Diderot<br> Paris 7
 | fondation={{Date|1|janvier|1971}}
  <ref name="décret de création">{{Légifrance|base=JORF|numéro=MENS0000215D|.
 | type=[[Université en France|Université publique]] ...
 | budget= 158 Millions d'euros ...
 | ville=[[Paris]]
 | pays={{France}}
 | région=[[Île-de-France]]
 | campus= [[Campus Paris Rive Gauche|Paris Rive Gauche]],...
 | langue= [[Français]]
 | président=[[Christine Clerici]]
 ....
}}
```


Wikipédia : rendu



Université Paris Diderot Paris 7

Vous lisez un « bon article ».

L'**université Paris Diderot**[0 1] (nom officiel : **Paris-VII**)[1] est une université pluridisciplinaire située à Paris, et elle est une des sept universités de l'académie de Paris. Elle a été créée en 1971 à la suite de la scission de l'université de Paris. Depuis son déménagement en 2007 du campus de Jussieu, elle est principalement implantée dans le quartier de Paris Rive Gauche. L'université est membre du PRES *Sorbonne Paris Cité* depuis le 31 mars 2010[2].

Elle est spécialisée dans le domaine des sciences, des formations du domaine de la santé, des sciences humaines et sociales, ainsi que des arts, lettres et langues. Ses activités de recherche se concentrent autour de 102 équipes (dont près de 80 % associées à de grands organismes de recherche) et regroupent près de 2 300 doctorants et 2 000 enseignants-chercheurs. Elle forme près de 25 000 étudiants.

Parmi les personnalités liées à l'université, deux enseignants ont obtenu un prix Nobel, et deux autres ont exercé la fonction de ministre de l'Éducation nationale en France.

Sommaire [masquer]

1 Historique
- 1.1 Création de l'université
- 1.2 Débuts difficiles
- 1.3 Développements à partir des années 1990
- 1.4 Création du campus « Paris Rive Gauche »

2 Administration
- 2.1 Gouvernance
- 2.2 Présidences

3 Composantes
- 3.1 Structures d'enseignement
- 3.2 Écoles doctorales
- 3.3 Service commun de documentation

4 Implantations
- 4.1 Campus Paris Rive Gauche
- 4.2 Locaux tampons en attente de finalisation du campus Paris Rive Gauche
- 4.3 Sites hospitaliers et cliniques universitaires

5 Enseignements et recherche
- 5.1 Formations
- 5.2 Échanges internationaux
- 5.3 Recherche
  - 5.3.1 Laboratoires de recherche
  - 5.3.2 Scientométrie

6 Vie étudiante
- 6.1 Sociologie
- 6.2 Activités sportives

Université Paris Diderot Paris 7

Informations

Fondation	1er janvier 1971[1]
Type	Université publique (EPSCP)
Régime linguistique	Français
Budget	158 Millions d'euros (10 M€ de loyer/an)[1].

Localisation

Coordonnées	48° 49′ 47″ Nord 2° 22′ 51″ Est
Ville	Paris
Pays	France
Région	Île-de-France
Campus	Paris Rive Gauche, Campus de Jussieu, Fontainebleau

Direction

Les expressions régulières

- Les expressions régulières définissent des ensembles de mots (aussi appelés *des langages*).
- 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 régulières (classiques)

- Donnée : un alphabet A, c.-à-d. un ensemble fini de symboles
- On définit d'abord la *syntaxe* des expressions régulières : Définition inductive de l'ensemble Reg.
- Puis on définit une *sémantique*, à l'aide d'une fonction *récursive* $\mathcal{L}(\cdot)$: Reg $\to A^*$

Syntaxe des Expressions Régulières

Définition inductive de l'ensemble Reg :

- Pour tout symbole $a \in A$: $a \in$ Reg
- $\epsilon \in$ Reg.
- Si $r_1, r_2 \in$ Reg, alors $r_1 r_2 \in$ Reg
- Si $r_1, r_2 \in$ Reg, alors $r_1 \mid r_2 \in$ Reg
- Si $r \in$ Reg, alors $(r) \in$ Reg
- Si $r \in$ Reg, alors $r^* \in$ Reg
- C'est tout.

ATTENTION

- Nous écrivons | pour l'union, pas + comme c'était fait dans le cours *Langages et Automates du S3*.
- Raison : c'est la convention utilisée par presque tous les outils informatiques qui travaillent avec les expressions régulières.
- Autre raison : nous allons utiliser un peu plus tard le symbole + pour autre chose.

Sémantique des Expressions Régulières

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

Exemples d'expressions régulières

- Nous choisissons pour l'exemple $A = \{a, b, c, d\}$
- $\mathcal{L}(abc \mid bcd) = \{abc, bcd\}$
- $\mathcal{L}(aa(b \mid c)dd) = \{aabdd, aacdd\}$
- $\mathcal{L}((a \mid b \mid c \mid d)*)$: l'ensemble de tous les mots sur A
- $\mathcal{L}((a \mid c)*)$: l'ensemble de tous le 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 régulières

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

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

Pourquoi des expressions régulières alors ?

- Expressivité limitée, mais ...
- On sait faire plein de choses avec, par exemple :
  - Décider appartenance d'un mot au langage
  - Décider vide, universalité
  - Calculer le complément par rapport à A^*
  - Calculer l'intersection

Sucre Syntaxique

- Des extensions de syntaxe qui sont pratiques, mais pas essentielles
- $r?$: soit le mode vide, soit un mot dans $\mathcal{L}(r)$
  Abréviation pour $\epsilon \mid r$
- $r+$: une séquence *non-vide* de de mots dans $\mathcal{L}(r)$
  Abréviation pour $rr*$
- $r\{n, m\}$ pour $n, m \in \mathbb{N}$: une séquence de i mots dans $\mathcal{L}(r)$, où $n \leq i \leq m$
  C'est aussi une abréviation ...

Systèmes d'expressions régulières

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

$$\begin{aligned} r_1 &= (a|b)*c \\ r_2 &= r_1+ \\ r_3 &= r_1 e r_2 \end{aligned}$$

- C'est simplement un raccourci :

$$\begin{aligned} r_1 &= (a|b)*c \\ r_2 &= ((a|b)*c)+ \\ r_3 &= (a|b)*ce((a|b)*c)+ \end{aligned}$$

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

Exemple : syntaxe lexicale des langages de programmation

- Décrit les règles d'écriture pour les mots élémentaires d'un langage informatique.
- Exemples de classes de mots élémentaires : les noms des variables autorisés, les constantes entières, les différents mots clefs, ...
- La syntaxe lexicale est seulement une partie de la syntaxe d'un langage informatique : il manque par exemple toutes les règles de bonne *structure* du programme (parenthèses, ...)

Exemple : Identificateurs

- $min = a \mid b \mid c \mid \ldots \mid z$
- $maj = A \mid B \mid C \mid \ldots \mid Z$
- $chiffre = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- $lettre = min \mid maj$
- $ident = lettre(lettre \mid chiffre \mid \_)*$

Raccourci : classes de caractères

- $[abc]$ comme raccourci pour $a \mid b \mid c$
- Si on a un ordre sur l'alphabet, comme pour UNICODE, on peut aussi permettre :

$$\begin{aligned} min &= [a..z] \\ lettre &= [a..zA..Z] \end{aligned}$$

- Le symbole . (un point) désigne n'importe quel symbole de l'alphabet.

Classe de caractères par exclusion

- Certains outils permettent aussi de définir des classes de caractères par *exclusion* :

$$r = [\hat{}\_\$]$$

  pour la classe de tous les caractères de l'alphabet *sauf* les deux caractères souligné et dollar.
- C'est le symbole ^ qui indique qu'il s'agit d'une exclusion.

Exemple : Chaînes de caractères

- Version simple : chaînes qui ne contiennent pas "

$$\begin{aligned} \textit{nonquote} &= [\tilde{}"] \\ \textit{string} &= "\textit{nonquote}*" \end{aligned}$$

- Permet par exemple :
  - "bonjour"
  - ""
- Ne permet pas :
  - "une chaîne sans fin
  - "abc"defg"

Exemple : Chaînes de caractères

- Plus sophistiqué : avec un caractère d'échappement \

$$\begin{aligned} \textit{nonspecial} &= [\tilde{}"\backslash] \\ \textit{string} &= "(\textit{nonspecial} \mid \backslash.)*" \end{aligned}$$

- Permet par exemple :
  - "abc\ndef\"ghi"
  - "\\\\"
- Ne permet pas :
  - "abc"def"
  - "abcd\"

Exemple : l'outil egrep

- Vu en S1 : outil `grep`, pour chercher un mot dans un fichier
- En fait, l'outil `grep` accepte aussi des expressions régulières comme motif de recherche.
- Malheureusement, il y a des systèmes de notations incompatibles pour ces expressions régulières : la page man de `grep` les appelle des *expressions régulières simples* et des *expressions régulières étendues* (angl : *basic regular expressions* et *extended regular expressions*).

Pourquoi deux systèmes de notion ?

- En théorie le monde est simple : les symboles d'un alphabet s'appellent a, b, c, etc., et ne risquent pas d'être confondus avec les symboles spéciaux comme (,), $+$, $*$, etc.
- Si on veut traiter des vrais langages informatique on a affaire à des alphabets qui contiennent tous les caractères spéciaux.
- Comment se débrouiller pour distinguer un symbole lettre x du symbole spécial x, tout en permettant des expressions aussi simples et lisibles que possible ?
- Il y a (au moins) deux solutions incompatibles à ce problème.

Les Expressions Régulières Étendues de UNIX

- Correspondent plus au moins aux expressions régulières comme définies ici :
- | pour union, opérateurs postfix $*$, $+$, ?, $\{n, m\}$
- Quelques classes de caractères définies, comme `[[:digit:]]` (chiffres), `[[:upper:]]` (lettres en majuscules), `[[:lower:]]` (minuscules), etc.
- Utiliser le symbole d'échappement \ pour transformer un caractère spécial en lettre normale.
- Voir la page man, et le TP1, pour plus de détails

Utilisation de egrep

- Utiliser la commande `egrep` pour chercher avec une expression régulière dans un fichier (ou `grep -E`)
- Attention aux symboles spéciaux de la shell : Normalement on veut mettre l'expression régulière entre des apostrophes ' et '.
- Exemple : `egrep 'Odette|Gilberte' swann.txt`

Sémantique

- Dans le cas de `egrep` c'est facile : pour savoir si une ligne contient un mot reconnu par r il faut tester si la ligne entière est reconnue par $.*r.*$.
- C'est plus délicat quand on veut *extraire* le mot reconnu.
- Raison : il se peut qu'une ligne contient plusieurs sous-mots qui sont reconnues par l'expression régulière. Laquelle choisir ?

Exemple

- Expression Régulière : $(cd)*|(ab)*$
- Ligne de texte : *eeabeecdcdee*
- Réponses possibles :
  - *ab*
  - *cd*
  - *cdcd*

Désambiguïser la sémantique

- C'est à chaque outil de spécifier exactement comment il cherche un mot reconnu par une expression régulière :
  - Chercher le sous-mot reconnu qui commence le plus à gauche sur la ligne ? Ou plutôt essayer les alternatives de l'expression régulière dans l'ordre ?
  - Préférez le sous-mot reconnu le plus court, ou le plus long ?
- *Normalement :* on veut le mot dont le début est le plus à gauche, et puis le plus long possible.