Conception et pratique de l'algorithmique

Module CPA

Philippe Trébuchet

Université Pierre et Marie Curie

Second semestre 2014-2015
Version du 10 janvier 2015

Première partie I

Les Expressions régulières

Plan

Les expressions régulières : généralités

- Langage de Description de chaînes de caractères.
- Utilisé pour reconnaitre un motif dans un un corpus de texte
- plusieurs dialectes différents

Rappels : Les ERE Posix : Le tronc commun avec les autres dialectes

Un caractère non spécial se reconnait lui même.

- `.` signifie n'importe quel caractère.
- `[`$c_1 c_2 c_3$`...]` signifie *un* caractère parmi c_1, c_2 ou c_3
- `[^`$c_1 c_2 c_3$`...]` signifie *un* caractère qui n'est pas c_1, c_2 ou c_3
- `\ char_special` signifie le caractère spécial est considéré comme un caractère normal (\ enlève la signification spéciale d'un caractère spécial).
- ^signifie début de ligne
- `$` signifie fin de ligne.

- Dans un bloc crochet `[]` `$` et `^` (en dehors du premier caractère. perdent leur signification spéciale !

Les ERE Posix (Suite)

- les caractères spéciaux des ERE sont : `. + * { } ( ) ^ $ [ ] | ?`
- `()` définition de bloc parenthésé. pas de notion de rappel du i-ème bloc parenthésé
- `(ERE1|ERE2)` alternance : `ERE1` ou `ERE2`.

multiplicateur d'occurences

- `ERE{n}`
- `ERE{m,n}`
- `ERE{m,}`
- `ERE{0,n}`
- `ERE+` signifie : au moins une fois l'`ERE`
- `ERE?` zéro ou une fois l'`ERE`

Exemples d'ERE

- reconnaitre une chaine formée de `a` :
  - idem BRE `^a*$`
- reconnaitre une chaine contenant entre 3 et 5 `a` consecutifs.
  - `a{3,5}` (attention en ERE `{` et `}` sont des caractères spéciaux)
- reconnaitre une chaine contenant la chaine `toto` répétée entre 3 et 5 fois d'a filée
  - `(toto){3,5}`
- reconnaitre une chaine commencant soit la chaine `toto`, soit la chaine `tutu`
  - `^(toto|tutu)` Les parenthèses sont obligatoires

Extensions GNU

Les outils GNU définissent en plus de RE précédentes un certain nombre d'extensions pratiques :

- `\b` match une chaine vide à une frontiere de mot.
- `\<` et `\>` matchent une chaine vide en debut et fin de mot.
- `\w` match un word constituent
- `\s` est un synonyme pour `[[:space:]]`

Des outils de filtrage puissants

- `grep` `Get Regular Expression and Print`
- Synopsis : `grep option BRE [noms de fichiers]`
- Si aucun nom de fichier n'est donné `stdin` est utilisé.
  - `-l` affiche le nom du fichier au lieu de la ligne.
  - `-E` utilise de ERE au lieu de BRE.
  - `-F` utilise des chaînes de caractères pour rechercher.
  - `-c` compte le nombre de matchs.
  - `-e` plusieurs motifs.
  - `-i` case insensitif.
  - `-q` quiet.
  - `-s` supprime les messages d'erreur.
  - `-v` inverse la sélection.

Exemples de `grep`

- Afficher les lignes contenant la chaine `toto` :
  - `grep -f toto des_noms_de_fichiers`
- Afficher le nom des fichiers contenant des lignes commencant par `toto`
  - `grep -l '^toto' des_noms_de_fichiers`
- Tester si le fichier `fic` contient bien la chaine de caractere `string` en majuscule ou minuscule et si oui afficher `ok`
  - `grep -iqs 'string' fic && echo ok`
- On peut aussi utiliser `grep` en conjonction avec l'option `exec` de find !
  - `find /etc/ -exec grep -il 'printer' {}\;`

Rappel Lexing/Parsing : Lexeur

- c'est un logiciel qui va reconnaitre dans le texte une suite de caractères et va leur associer un jeton.
- est essentiellment un moteur d'inférence d'expressions régulières
- Les jetons (token) seront transmis au parser qui lui executera la grammaire.

(f)Lex

- Logiciel ancestral de génération de lexeurs

```
%{
        Code Verbatim
%}
 definition

%%
ERE {ACTION}
<context>ERE {ACTION}

%%
Code
```

Lexeur à la main

```perl
#!/usr/bin/perl
sub lexer
{
	my $string=shift;
	return sub {
		return "LPAREN" if ($string=~/\G\(/cg);
		return "RPAREN" if ($string=~/\G\)/cg);
		return "PIPE" if ($string=~/\G\|/cg);
		return "LBRACKET" if ($string=~/\G\[/cg);
		return "RBRACKET" if ($string=~/\G\]/cg);
		return "POINT" if ($string=~/\G\./cg);
		return "STAR" if ($string=~/\G\*/cg);
		return "PLUS" if ($string=~/\G\+/cg);
		return "QUESTIONMARK" if ($string=~/\G\?/cg);
		return "CHAR" if ($string=~/\G\w/cg);
		return "0";
	}
}
$lex=lexer($ARGV[0]);
print "$toto " while ($toto=$lex->());
```

Rappel Lexing/Parsing : Parseur

- logiciel qui interprete le flux de token selon une grammaire.
- Parseur reccurssif descendant
- Parseur ascendant
- Production d'un *arbre de syntaxe* (AST).

Yacc

- Logiciel ancestral de generation de parseur LALR (Ascendant)

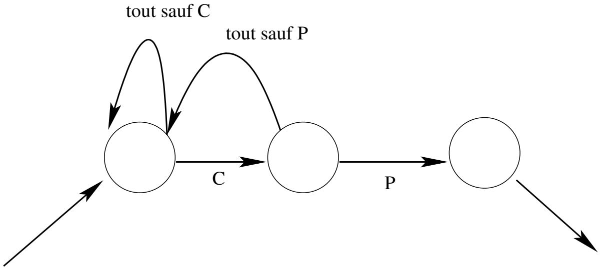
```
%{
  code verbatim
%}
 definition de token
 definition de macros

%%

 grammaire

%%

 code verbatim
```

```
sub parser {
  my $tmp;
  my $lex=lexer(shift);
  my $parse=sub {
    my $res="";
    my $token=$lex->();
    if ($token=~/LBRACKET/)
    {
      $res.="[";
      $res.=$token until (($token=$lex->())=~"RBRACKET");
      return $res."]";
    }
    if ($token eq /LPAREN/)
    {
      $res.="(".$lex->() until ($lex->() eq "RPAREN");
      return $res.")";
    }
#           if($token=~/
    print "aaaaa";
    return $res;
    };
```

Complexité

- Lexing : Linéaire !
- Parsing : Dépend beaucoup de la grammaire
  - de $\mathcal{O}(n^3)$.
  - à exponentielle.

Reconnaitre une chaine de caractères



Deuxième partie II

Les automates

automate

Définition

Un automate fini sur un alphabet $\mathcal{A}$ est la donnée de :

- un ensemble d'états E
- une fonction de transition f
- un état initial i
- un ensemble d'etats finaux O

Accepté ou pas ?

- Un mot *m* est accepté par l'automate si et seulement si en partant de l'etat initial et en ne suivant que les transitions libellées par les lettres de *m* on peut aboutir à un etat final.

Définition

L'ensemble des mots accéptés par un automate s'appelle le langage accépté.

Définition

Deux automates sont équivalents s'ils reconnaissent le même langage.

Automate déterministe vs non déterministe

definition

Si pour tout état e il n'y qu'au plus une transition étiquettée par une lettre a de l'alphabet, on dit que l'automate est deterministe. On dit que l'automate est non deterministe sinon.

On dit qu'un automate est complet si pour tout état e et toute lettre a, il existe au moins une transition qui part de e et qui est etiquetté par la lettre a.

ε-transition

- une transition est une ε-transition si elle n'est etiquettée par aucune lettre.

Un automate avec epsilon transition est un automate non deterministe.

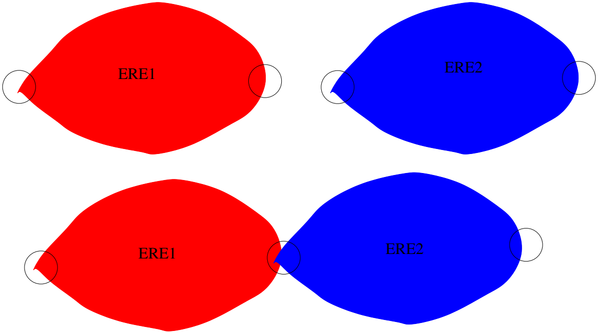
algorithme de reconnaissance

- Pour un automate fini déterministe il suffit de suivre les transitions. Le coût d'un match est linéaire en la taille de l'entrée.
- Pour un automate non déterministe, il faut explorer toutes les possibilités jusqu'à trouver un etat acceptant ou epuiser toutes les possibilités. Le coût peut etre exponentiel !

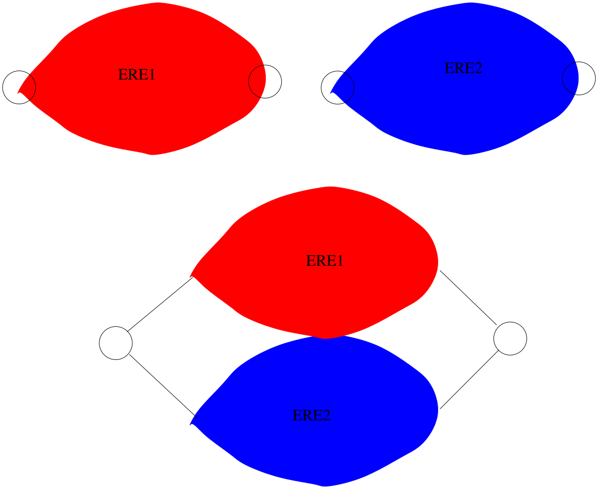
La construction de Thompson

ERE1ERE2	On concatene
ERE*	une epsilon transition vers le debut de l'ERE
ERE+	ERE concaténé à ERE*
(ERE1—ERE2)	un état + 2 epsilon transitions vers ERE1 et ERE2
ERE ?	une epsilon transition vers la suite ou vers ERE

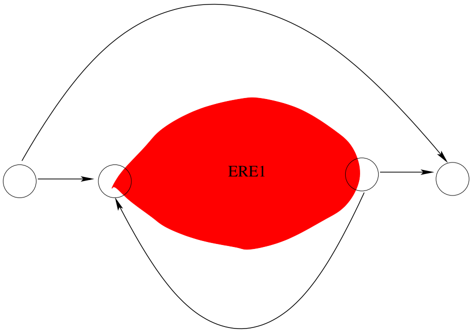
Concatenation



Union



Etoile



Propriétés

Propriété

Si l'expression régulière est formée de n symbols (lettre + opérateurs), l'automate produit a $2n$ états.

Propriété

La traduction d'une expression régulière en automate est linéaire en la taille de l'expression.

Propriété

Chaque état de l'automate produit est la source (resp. la destination) d'une ou deux transitions (sauf les états initiaux et finaux).

Propriété

La construction précédente ne tient pas compte de l'associativité des opérateurs.

Structures de données

- les états peuvent etre numérotés par des entiers
- la fonction de transition peut etre représenté comme une matrice
  - d'entiers (deterministe)
  - de liste d'entiers (non deterministe)

Propriétés des automates

- Soit L un langage reconnaissable, il existe un automatique deterministe le reconnaissant.
- Il existe des langages non reconnaissables.
- si L et L' sont des langages reconnaissables alors $LŁ'$ l'est aussi

Lemme de l'etoile

Si L est un langage reconnaissable, alors il existe un entier N tel que tout mot m de taille supérieure à N peut se factoriser en trois sous mots u, v, w tels que, v est non vide, $m = uvw$ et $\forall n \in \mathcal{N},\ uv^n w \in L$.

Opérations sur les automates

- Si on inverse les flèches d'un automate et qu'on échange les états initiaux et finaux on a un automate qui reconnait le langage miroir.
- Il existe plusieurs automates pour le même langage.
- un automate complet est un automate pour lequel, pour chaque état e et pour chaque lettre l il existe une transistion partant de e et étiquetée par l.

Déterminisation

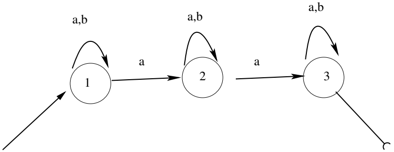
Proposition

Tout automate A est equivalent à un automate D complet et déterministe. Si A est fini avec n états, alors D peut être construit avec au plus 2^n états.

algorithme

- input : $A = (E, Q, I, F)$ un automate *sans epsilon transitions*
- output : B un automate fini deterministe equivalent à A.
- $Todo = I$, $E_B = i$, $Q_B = \emptyset$ $I_B = I$
- Tq Todo est non vide faire
  - $S = pop(Todo)$
  - pour chaque lettre l
    - Soit $E = e_1, \ldots, e_k$ l'ensemble des états accessibles depuis S par une transition etiquetée par l
    - si E n'est pas deja dans E_B
      ajouter E à E_B
    - ajouter une transition de S à E dans Q_B
- F_B est égal à l'ensemble des états de Q_B qui contiennent un état final de F.
- retourner $B = (E_B, Q_B, I_B, F_B)$

Exemple



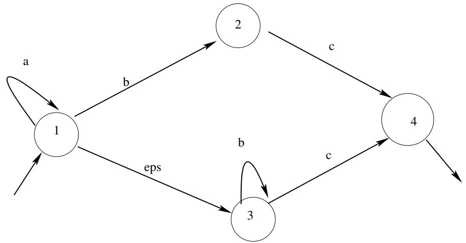
ε-transitions

Proposition

Etant donné un automate fini avec *varepsilon*-transitions il existe un automate fini sans *varepsilon*-transitions reconnaissant le même langage.

- L'algorithme fonctionne en remplaçant les chemins de longueur 1 commençant par une *varepsilon*-transitions par une nouvelle transition.

exemple



Minimisation

- L'automate deterministe construit avant est tres tres tres gros.
- il existe des automates plus petit.
- il existe plusieurs algorithmes pour calculer le plus petit automate équivalent.

Etats indistingables

Définition

Soit A un automate, deux etats x et y de A sont dits indistigables si et seulement si pour tout mot m tq xm est accepté alors ym l'est aussi et réciproquement.

Brzozowski

Définintion

Le transposé d'un automate est l'automate obtenu en inversant les transitions.

- L'automate transposé reconnait le langage miroir
- l'automate transposé peut avoir des états inaccessibles

Brzozowwski

Proposition

Soit A un automate deterministe accessible. Le determinisé du transposé de A est minimal.

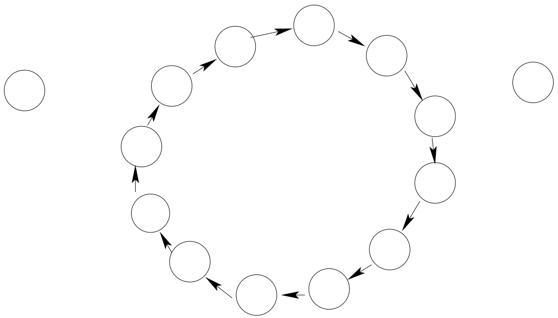
algo

Calculer $det(Trans(det(trans(A))))$

La construction de Moore

- On met les états finaux d'un coté et les autres de l'autre.
- On sature par la relation d'equivalence de distingabilité.
- Complexité : $\mathcal{O}(n^2)$.

Contre exemple



La construction de Hopcroft

Definition

- Soit E un ensemble d'etats et a une lettre on note $a^{-1}E$ l'essemble des q tels que une transition de q etiquetée par a amène dans E.
- Soit B un essemble d'état on dit que B est scindé par la pair (E, a) Si :
  - $B_1 = B \cap a^{-1}P$
  - $B_2 = B \backslash B_1$

  Sont tous les deux non vides. Sinon B est dit stable par (P, a).

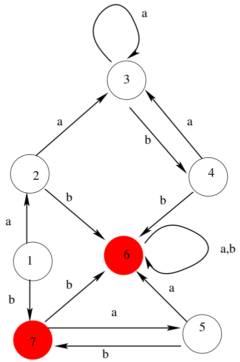
Principe de l'algorithme

- On part d'une partition en deux etat Finaux/Autres
- On maintient une liste de Paires scindantes
- Si une paire scinde un etat E en E_1, E_2 :
  - on remplace l'etat E par les etats E_1 et E_2
  - Si la paire (E, a) etait scindante on la remplace par les deux paires (E_1, a) et (E_2, a).
  - Sinon on choisi *le plus petit* de B_1 et B_2 et on rajoute les paires (B_i, a) comme paire scindante. pour toute lettre.
  - On s'arrete quand il ne reste plus de paires scindantes.

La construction de Hopcroft

- $L = \emptyset$
- Si $|F| < |Q/F|$ alors
  - $C_0 = Q/F$, $C_1 = F$ Rajouter C_1 à L.
- sinon
  - $C_1 = Q/F$, $C_0 = F$ Rajouter C_1 à L.
- $P = \{C_0, C_1\}$
- Tant que $L \neq \emptyset$ faire
  - Extraire C de L
  - Scinder chaque ensemble C_i construit jusqu'à présent par C
  - Ajouter à L la plus petite coupure

exemple



Complexité

- complexité en $\mathcal{O}(nlog(n))$
- en pratique la constante est grosse
- pour des tailles usuelles on utilise plus volontier l'algorithme de Moore.

Definition

Un automate généralisé est une automate dont les transistions sont étiquettées par des langages rationnels.

Propriétés

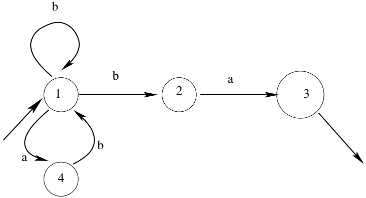
Le langage reconnu par un automate généralisé est un langage rationnel.

La construction de Brzozowski et McCluskey

Pour traduire un automate en expression :

- On commence par rajouter un unique etat initial et un unique etat final à l'automate
- tant qu'il reste un etat autre que ces deux là :
  - choisir un etat et eliminer les transisitions entrante et sortantes

Exemple



Troisième partie III

automates à pile

definition

Un automate à pile est un sextuplet :

- un alphabet de transition
- un ensemble d'etats
- un etat initial
- un ensemble d'etats finaux
- un alphabet de pile
- un ensemble de transitions

Les transitions sont données par des triplets `([i,P],a,[j,Q])` signifiant ;

- on passe de l'etat *i* à l'etat *j*
- on depile P puis on empile Q

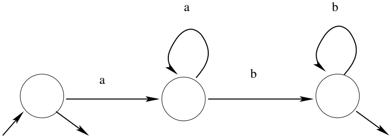
Accéptation

definition

On accepte un mot si et seulement si en partant de pile vide et en suivant les transitions données par le mot on abouti dans un etat acceptant ET avec une pile vide.

Exemple : $a^n b^n$

Un seul symbole de pile !



Equivalence : Automate à pile/ Grammaire algébrique

Théorème

Soit G un grammaire algébrique. Le langage L_G engendré par G peut etre reconnu par un automate à pile A_p.

- c'est la construction faite par Yacc !
- il existe une construction inverse !

- etats : une seul et unique s !
- alphabet de pile : les terminaux et les variables
- Pour chaque production de la forme $A \rightarrow B$ on ajoute une ε-transition $[s, A] \rightarrow [s, M]$
- Ajout des transitions pour depiler les symboles terminaux.

Langage algébrique

Définition

Un langage reconnu par un automate à pile est appellé langage algébrique.

Proposition

- l'intersection entre un langage algébrique et un langage regulier est algébrique.
- Il existe des langages algébriques dont l'intersection n"est pas algébrique.
- Les langages algébrique ne sont pas fermés par complémentation.

Lemme de l’etoile pour les langages algébriques

SI L est un langage engendré par une grammaire G alors il existe un entier N tel que tout mot de L de longueur supérieure à N peut se factoriser sous la forme $uxzyv$ tels que :

- x et y sont non vides
- $ux^nzy^nv \in L$ pout tout $n \in \mathbb{N}$.

Quatrième partie IV

Machine de Turing

Definition

Une machine de Turing sera définie par la donné de :

- un alphabet fini.
- un ruban infini découpé en cases, chaque case pouvant contenir exactement un symbol de l'alphabet fini.
- d'une tête de lecture écriture pouvant se déplacer d'une case a la fois sur le ruban, lire la case où elle se trouve, écrire un symbole dans la case où elle se trouve et changer d'état. La tête ne peut prendre qu'un nombre fini d'etats.
- un état initial pour la tête et un eťat final acceptant signifiant que le calcul s'est bien passé.

On dit qu'une machine de Turing est déterministe si étant donné un état de la tête et un symbole dans la case lue, on n'a qu'une seule action possible. Elle est non déterministe sinon.

Propriétés

- tout ce qu'on peut calculer avec une machine de turing à n rubans est calculable avc un machine à un seul ruban.
- tout ce qui peut se calculer avec un ruban infini dans les deux sens peut aussi être calculé avec un ruban infini dans une seule direction.

Problèmes NP

- un probleme qui est accépté par une machine de Turing deterministe en in nombre polynomial d'operations appartient à la classe de complexité *P*.
- Un Problème est dit NP s'il est accepté en temps polynomial par une machine de Turing non déterministe

Completude

- une reduction polynomiale d'un probleme A à un problème B est un calcul polynomial qui transforme la resolution du probleme A en celle du probleme B.
- Un probleme que l'on peut reduire à un autre est plus facile a resoudre.
- Les problemes les plus durs d'une classe sont appelés *complet*.

Completude

- Tous les problemes complets sont aussi durs les un que les autres.

Propriétés

On ne connait pas d'algorithme mieux qu'exponentiel pour resoudre un probleme NP-Complet.

Un probleme NP-Complet : SAT

Enoncé

Etant donné une formule logique sous forme normale conjonctive $\bigvee_{i=1}^{n} v_i = v_1 \vee v_2 \vee \ldots \vee v_n$ existe t'il une valeur des varible qui la satisfasse.

SAT est NP-Complet

Considérons maintenant un problème de NP de taille n, i.e. il existe une machine de Turing non-déterministe à q états sur un alphabet à s lettres résolvant le problème en temps polynomial, soit $P(n)$ le nombre d'opérations nécessaires. Posons

- pour $i \in [0, P(n)],\ j \in [0, q-1]$ on introduit $Q_{i,j}$ qui voudra dire qu'après la i-ième opération la machine se retrouve dans l'état j.
- pour $i \in [0, P(n)],\ j \in [-P(n), P(n)]$ et $k \in [1, s]$ on introduit $S_{i,j,k}$ qui voudra dire qu'après la i-ème opération la case numéro j contient la lettre a_k.
- Pour $i \in [0, P(n)],\ j \in [-P(n), P(n)]$ on introduit $T_{i,j}$ qui voudra dire qu'après la i-ème opération la machine lit la case j.

Demonstration

- clause stipulant qu'à l'étape i la machine de Turing est dans au moins un état
- a la i-ème opération la machine ne peut pas être dans l'état j et dans l'état k a la fois
- clauses qui sont vraies si et seulement si chaque case de la bande dont l'indice est compris entre $-P(n)$ et $P(n)$ contient exactement un symbol de l'alphabet.
- 'a l'étape i, la machine est en train de lire une et une seule case de la bande.
- a l'étape 0 la machine est dans l'état 1 et est en train de lire la case 0.
- implication qu'à l'étape i si la machine est dans l'état k, qu'elle lit la case j dans laquelle se trouve le symbole l, alors elle passe dans un état $Q_{i+1,t(k,l)}$, où $t(k,l)$ est la fonction exprimant le nouvel état de la machine.
- implication qu'à l'étape i si la machine est dans l'état k, qu'elle lit la case j dans laquelle se trouve le symbole l, alors elle écrit le symbole $u(k,l)$ dans
- sous les même conditions la tête de la machine va à la case $j + d(k,l)$
- ...