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

```
#!/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
```

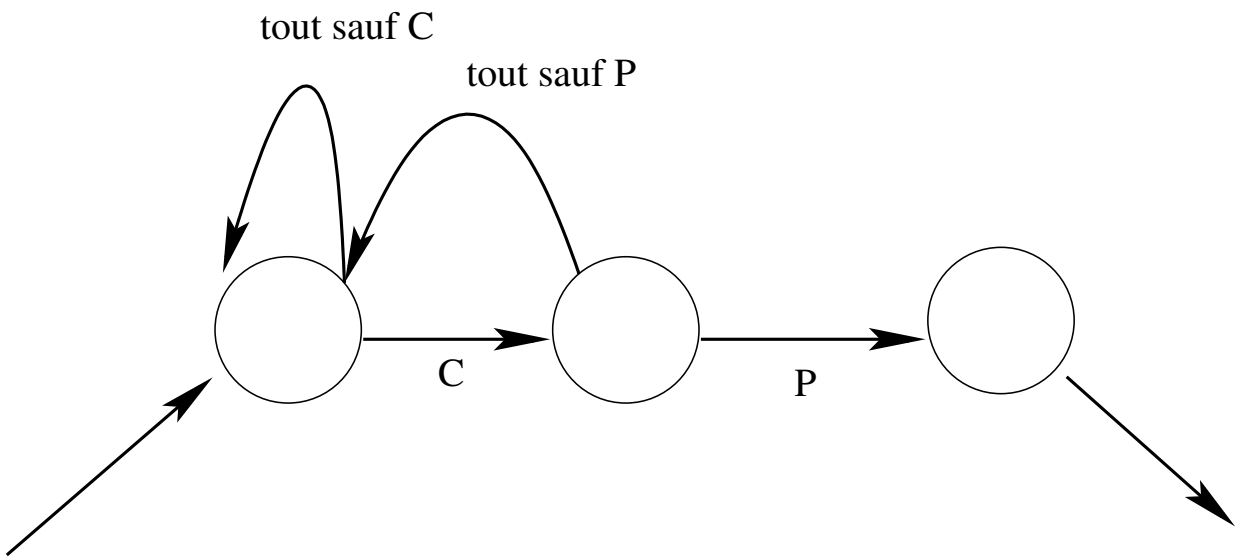
A la main

```
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



automate

Definition

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.

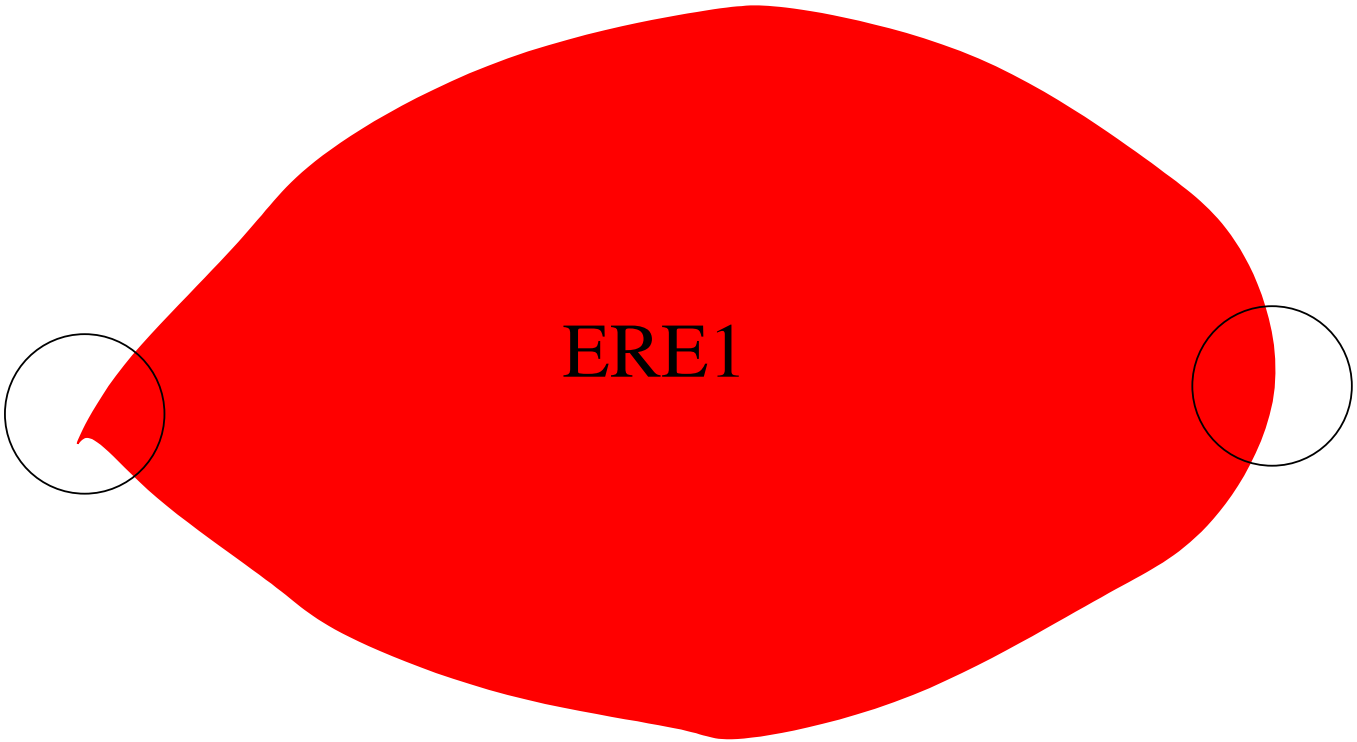
algorithme

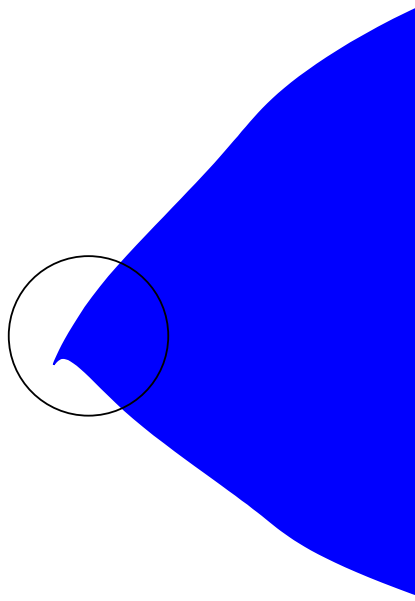
- Pour un automate fini déterministe il suffit de suivre les transitions. Le coût d'un match est linéaire en la taille de l'entrée.
- Pour un automate non déterministe, il faut explorer toutes les possibilités jusqu'à trouver un etat acceptant ou epuiser toutes les possibilités. Le coût peut etre exponentiel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

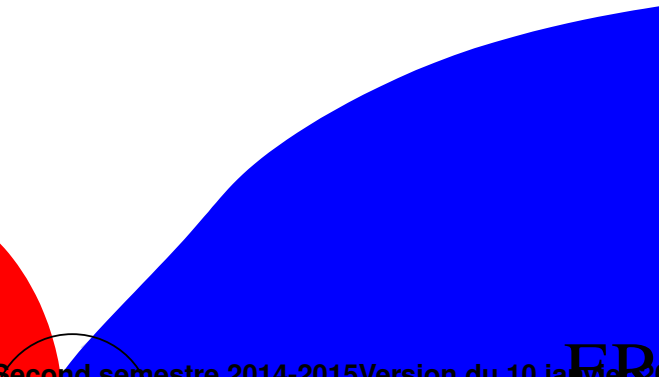
La construction de Thompson

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

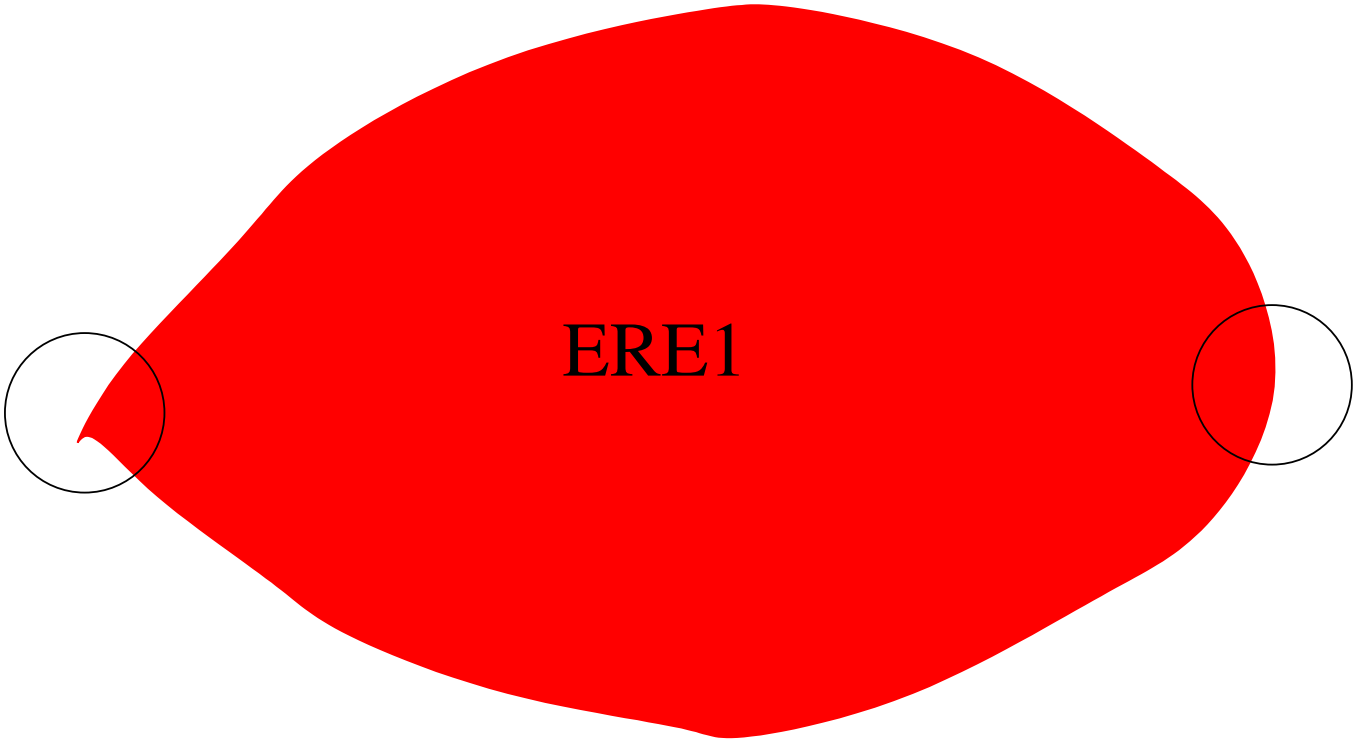
Concatenation

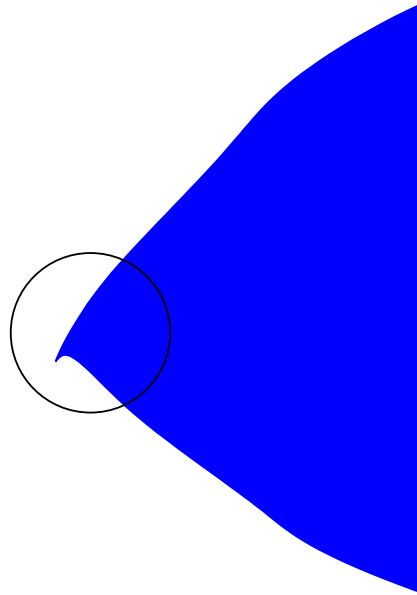




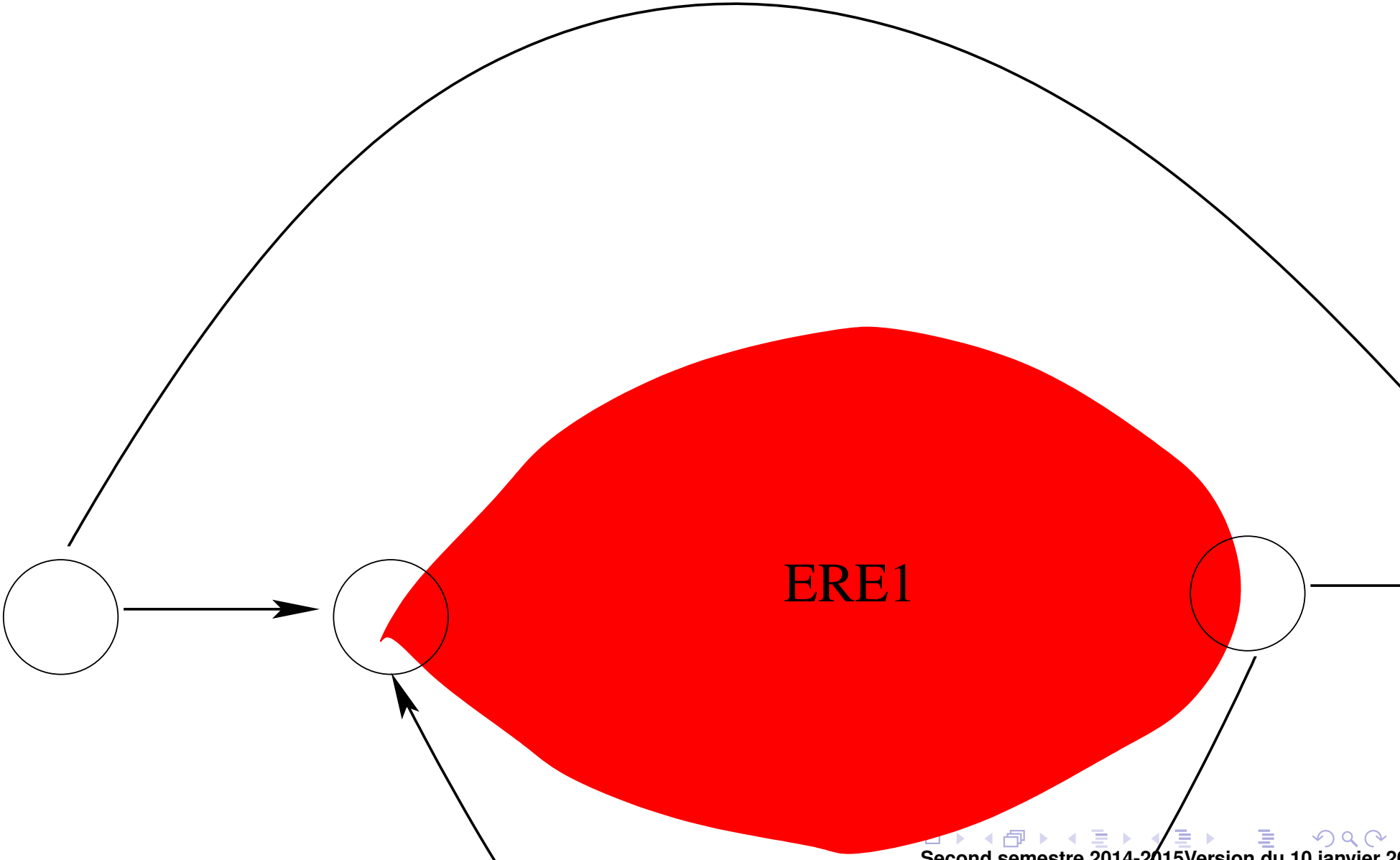


Union





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 etat 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
  - de liste d'entiers

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 $L Ł'$ 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 etre 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 $B = (E_B, Q_B, I_B, F_B)$

ε-transitions

- Modifier l'algorithme précédent pour clore les ensembles d'etats par epsilon transitions.
- on garde le même nombre d'etats maximal !

Minimisation

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

Browzoozozski

La construction de Hopcroft