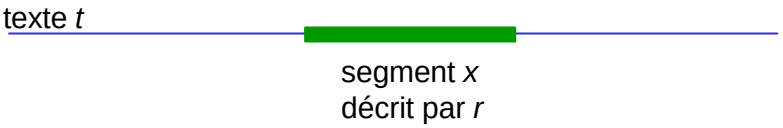
Reconnaissance de motifs

Problème

localiser les segments d'un texte décrits
par une expression rationnelle (régulière) *r*



Applications

édition : vi, emacs , sed, ed, **...**
recherche : grep, egrep , **...**
traduction : lex, sed , **...**
langages : awk, perl, **...**
compression : compress, gzip, **...**

GREP


```
unix > grep toto t.txt
```

produit les lignes de **`t.txt`**
qui contiennent le mot **`toto`**

Applications

recherche de fichiers par le contenu
contrôle du contenu d 'un fichier

Versions

`egrep, fgrep, ...`

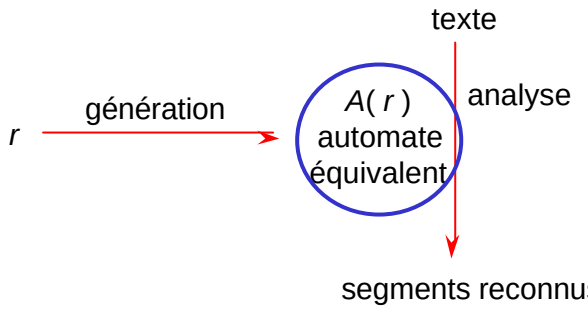
Algorithme de base

UMLV ©

Deux phases

1 génération d'un automate *A*(*r*)
2 analyse du texte avec *A*(*r*)

les phases peuvent être intégrées



Expression / langage

UMLV ©

Expression rationnelle *r*	Langage représenté *L*(*r*)
a, **b**, **…**	{**a**} , {**b**} , … **a**, **b** Î **A**
Æ, e	Æ , {mot vide}
[*u* , *v*	*L*(*u*) , *L*(*v*)]
(*u***)**	*L*(*u*)
u **+** *v*	*L*(*u*) È *L*(*v*)
uv	{ *xy* | *x* Î *L*(*u*), *y* Î *L*(*v*) }
*u******	{ $x_1x_2…x_k$ | *k* ³ 0, x_i Î *L*(*u*) }

`1(0+1)*0`	{écritures binaires des entiers pairs}
`(a*b)*a*`	{suites finies de **a** et **b**}
`(c*l)*c*f`	{fichiers textes}

Analyse des expressions

Une grammaire des expressions rationnelles

E ® *T* | *T* '+' *E*
T ® *F* | *F T*
F ® *S G*
G ® e | '*' *G*
S ® '**a**' | '**b**' | 'Æ' | 'e' | '(' *E* ')'

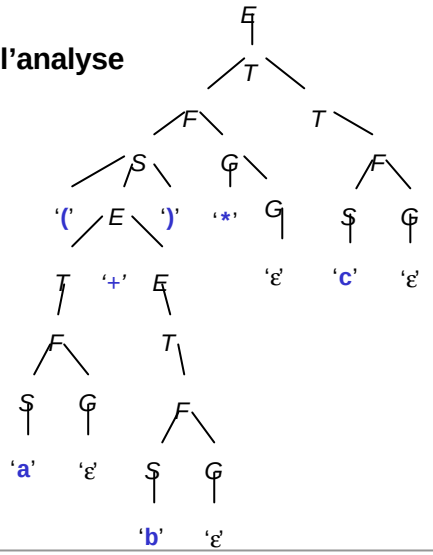
E expression, *T* terme
F facteur, *S* facteur simple

(a+b)*c {suites finies de **a** et **b** terminées par **c**}

Analyse de (a+b)*c

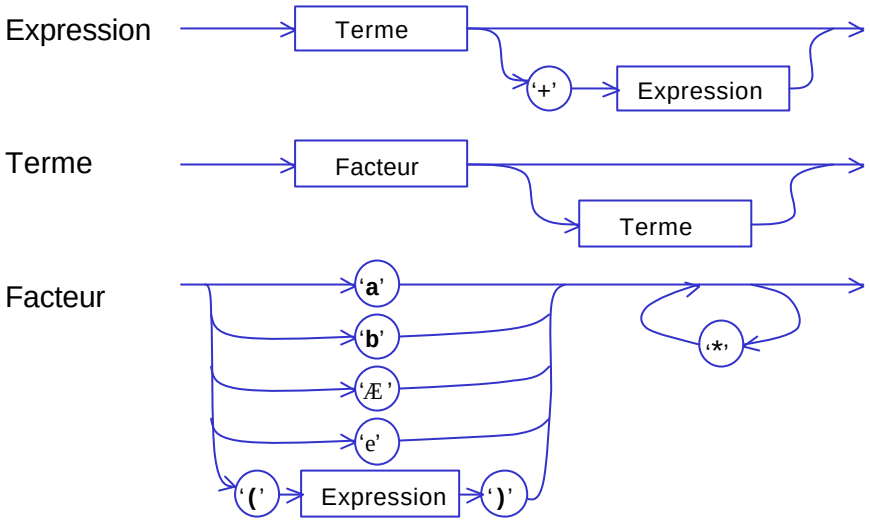
Grammaire
E ® *T* | *T* '+' *E*
T ® *F* | *F T*
F ® *S G*
G ® e | '*' *G*
S ® '**a**' | '**b**' | 'Æ' | 'e' | '(' *E* ')'

Arbre de l'analyse



Diagrammes syntaxiques

UMLV ©



Algorithme d'analyse

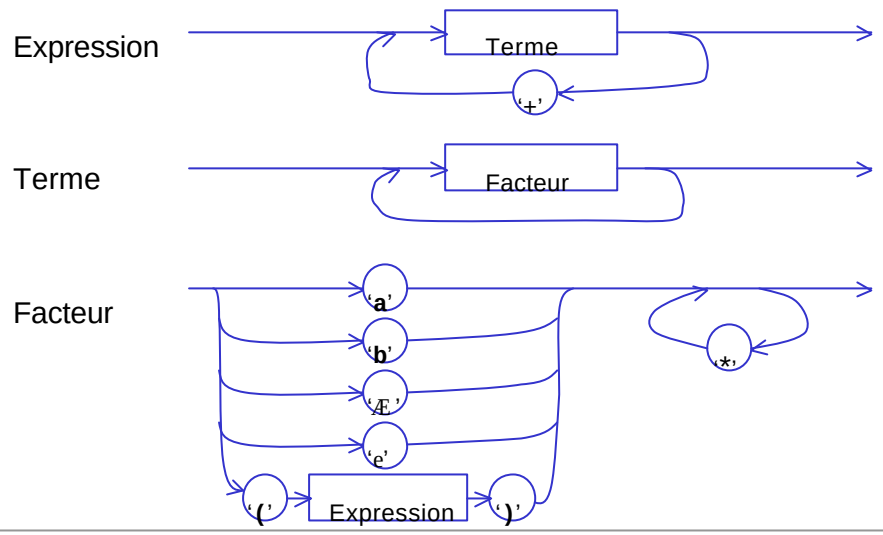
UMLV ©

```
caractère car;  /* caractère suivant, global */

Analyse(expression rationnelle r){
      car ¬ premier caractère de r ;
      Expression();
      si(car ≠ fin d'expression)
            erreur();
}
Expression(){
      Terme();
      si(car = '+'){
            car ¬ caractère suivant;
            Expression();
      }
}
Terme(){
      Facteur();
      si(car Î {'a', 'b', 'Æ', 'e', '('})
            Terme();
}
```

Algorithme d'analyse (suite)


```
Facteur(){
      si(car Î {'a', 'b', 'Æ', 'e'}){
            car ¬ caractère suivant;
      }sinon si(car = '('){
            car ¬ caractère suivant;
            Expression();
            si(car = ')')
                  car ¬ caractère suivant;
            sinon
                  erreur();
      }sinon
            erreur();
      tant que(car = '*')
            car ¬ caractère suivant;
}
```


Diagrammes syntaxiques (autre)



Algorithme d'analyse (autre)


```
caractère car;  /* caractère suivant, global */

Analyse(expression rationnelle r){
      car ¬ premier caractère de r ;
      Expression();
      si(car ≠ fin d'expression)
            erreur();
}
Expression(){
      Terme();
      tant que(car = '+'){
            car ¬ caractère suivant;
            Terme();
      }
}
Terme(){
      répéter
            Facteur();
      tant que(car Î {'a', 'b', 'Æ', 'e', '('})
}
```


Expression / automate

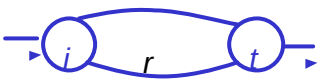
A(*r*) automate associé à l'expression rationnelle *r* sur l 'alphabet **A**

Invariants de la construction

A(*r*) possède un seul état initial *i*
un seul terminal *t*
aucune flèche n'entre dans *i*
aucune flèche ne sort de *t*

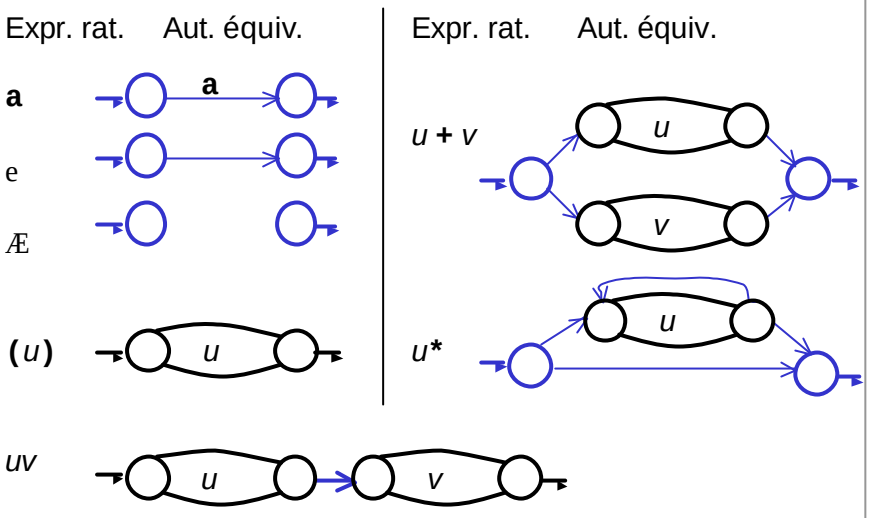
Graphique

A(*r*) :



Transformation

UMLV ©



Automate obtenu

UMLV ©

Proposition

A(*r*) reconnaît le langage *L*(*r*)

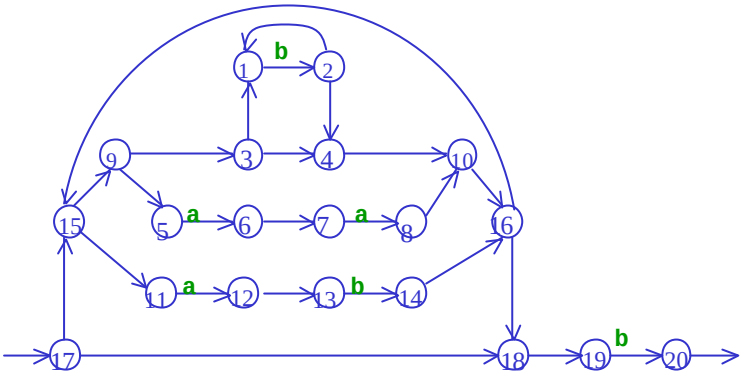
Propriétés supplémentaires

A(*r*) possède 2.|*r* | états, sans compter ‘(‘ ni ‘)’
de chaque état sortent au plus 2 flèches
si exactement 2, elles sont sans étiquette
sur chaque état arrivent au plus 2 flèches

Exemple

UMLV ©

Expression : **(b* + aa + ab)*b**



bab reconnu ?
non, mais infinité de chemins d'origine 17 et d'étiquette **bab**

Implantation adaptée

UMLV ©

état = indice
table des flèches

```
struct{
      caractère lettre;
      état suiv1, suiv2;
} Trans[MAX];
```

automate (identifié à 2 états)

```
struct{
      état initial;
      état terminal;
} automate;
```

Trans

p	'a'	p1	
q		q1	q2

initial = 17
terminal = 20

1	'b'	2	
2		1	4
3		1	4
4		10	
5	'a'	6	
6		7	
7	'a'	8	
8		18	
9		3	5
10		16	
11	'a'	12	
12		13	
13	'b'	14	
14		16	
15		9	11
16		15	18
17		15	18
18		19	
19	'b'	20	
20			

Algorithme de traduction

Production de l ’automate *A*(*r*)
associé à l ’expression rationnelle *r*

```
Analyse(expression rationnelle r){
      car ¬ premier caractère de r ;
      A ¬ Expression();
      si(car ≠ fin d’expression)
            erreur();
      sinon
            retour A;
}
```


Algorithme de traduction (suite)


```
Expression(){
      A ¬ Terme();
      tant que(car = ‘+’){
            car ¬ caractère suivant;
            B ¬ Terme();
            A ¬ Union(A,B);
      }
      retour A;
}
Terme(){
      A ¬ Facteur();
      tant que(car Î {‘a’, ‘b’, ‘Æ’, ‘e’, ‘(’}){
            B ¬ Facteur();
            A ¬ Produit(A,B);
      }
      retour A;
}
```

Algorithme de traduction (suite)


```
Facteur(){
      si(car Î {'a', 'b', 'Æ', 'e'}){
            A ¬ Automate-élémentaire(car);
            car ¬ caractère suivant;
      }sinon si(car = '('){
            car ¬ caractère suivant;
            A ¬ Expression();
            si(car = ')')
                  car ¬ caractère suivant;
            sinon
                  erreur();
      }sinon
            erreur();
      tant que(car = '*'){
            A ¬ Etoile(A);
            car ¬ caractère suivant;
      }
      retour A ;
}
```


Reconnaissance (avec automate non-déterministe)

Écrire une fonction

`booléen reconnaît( automate` *A*`, mot` *x* `)`

qui prend la valeur vraie ssi
il existe un chemin d'étiquette *x* depuis *A.initial* jusqu'à *A.terminal*

Programmation par

essais et erreurs (attention aux cycles)
programmation dynamique (mémorisation des états possibles)

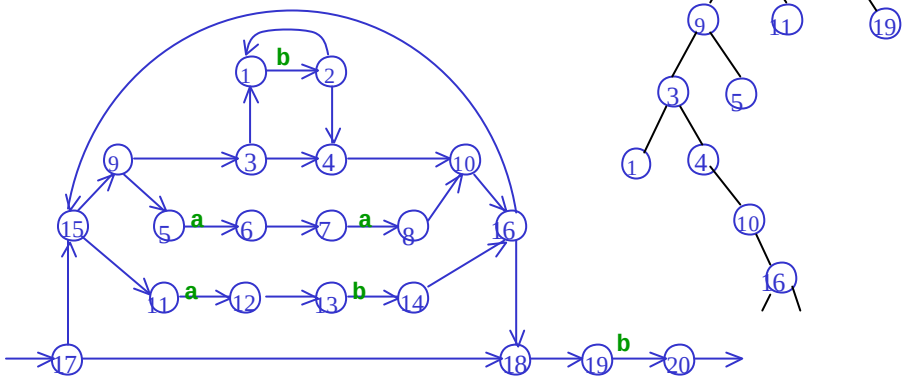
Clôture

clôture(*p*) = { états accessibles à partir *p*
par chemins sans étiquette }

UMLV ©

Clôture

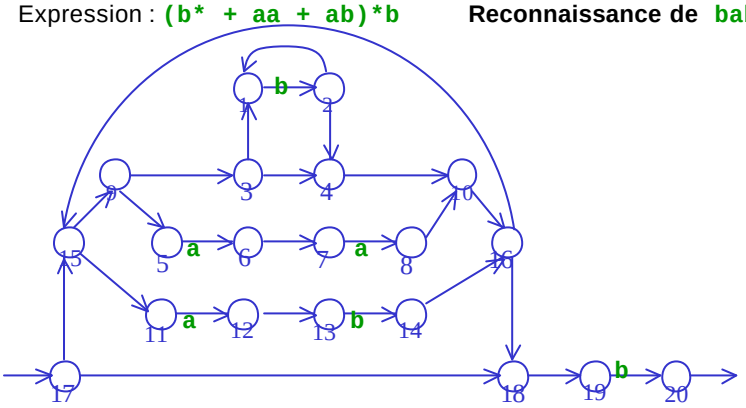
Clôture de 17 = {1,3,4,5,9,10,11,15,16,17,18,19}



UMLV ©

Reconnaissance

Expression : `(b* + aa + ab)*b` **Reconnaissance de** `bab`

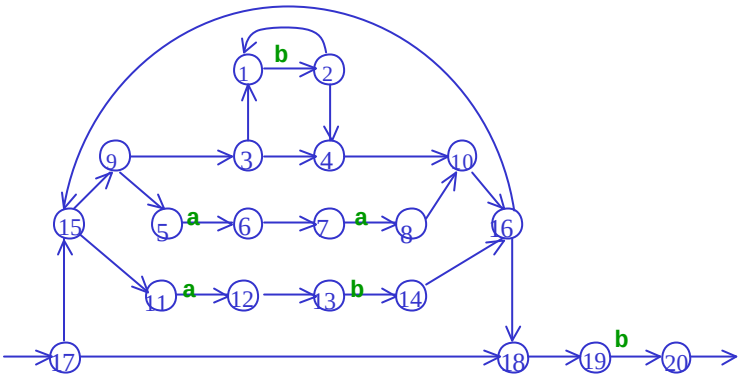


État initial 17
Clôture {1, 3, 4, 5, 9, 10, 11, 15, 16, 17, 18, 19}

UMLV ©

Reconnaissance (suite)

Expression : **(b* + aa + ab)*b** **Reconnaissance de bab**



{1, 3, 4, 5, 9, 10, 11, 15, 16, 17, 18, 19}

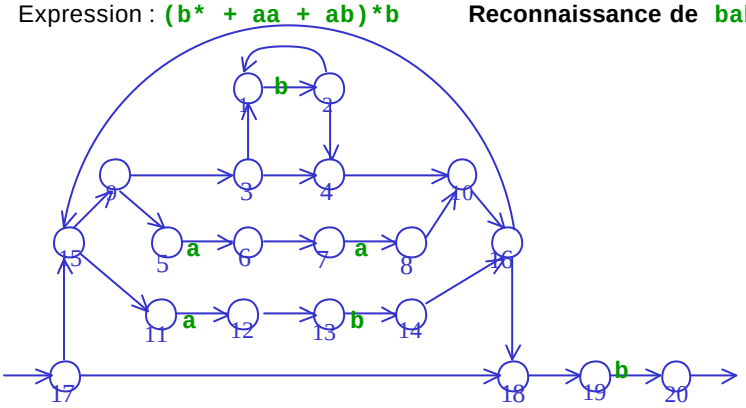
Transition par b {2, 20}

Clôture {1, 2, 3, 4, 5, 9, 10, 11, 15, 16, 18 , 19, 20}

UMLV ©

Reconnaissance (suite)

Expression : **(b* + aa + ab)*b** **Reconnaissance de bab**



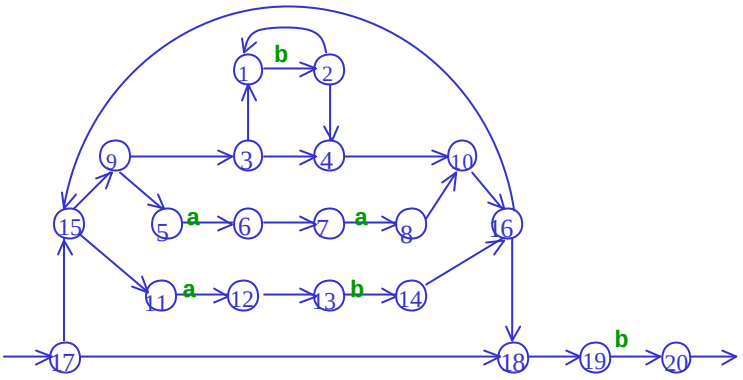
{1, 2, 3, 4, 5, 9, 10, 11, 15, 16, 18, 19, 20}

Transition par a {6, 12}

Clôture {6, 7, 12, 13}

Reconnaissance (suite)

Expression : `(b* + aa + ab)*b` **Reconnaissance de** `bab`



{6, 7, 12, 13}

Transition par `b` {14}

Clôture {1, 3, 4, 5, 9, 10, 11, 14, 15, 16, 18, 19}

`bab` **non reconnu car** 20 **non atteint**

Algorithme de reconnaissance

```
booléen reconnaît(automate A, mot x){
     ensemble E;
     E ¬ clôture(A.initial);
     tant que(non fin de x){
          car ¬ lettre suivante de x;
          E ¬ clôture(transition(E,car));
     }
     si(A.terminal Î E) retour(vrai);
     sinon retour(faux);
}
```

Lecture séquentielle du mot.
Mémorisation (tampon) d'une seule lettre de *x*.
Temps : O(nb états de *A* . | *x* |)
(si bonne gestion de l 'ensemble *E*)

Recherche de motif

UMLV ©

Motif défini par l'expression rationnelle *r*
A(*r*) automate associé à *r*

Motif dans le texte

texte *t*

segment *x*
reconnu par *A*(*r*), *x* Î *L*(*r*)

Principe de la recherche

comme reconnaissance avec :
ajout l'état initial à chaque position
arrêt de la recherche dès que l'état terminal est atteint

Algorithme de recherche

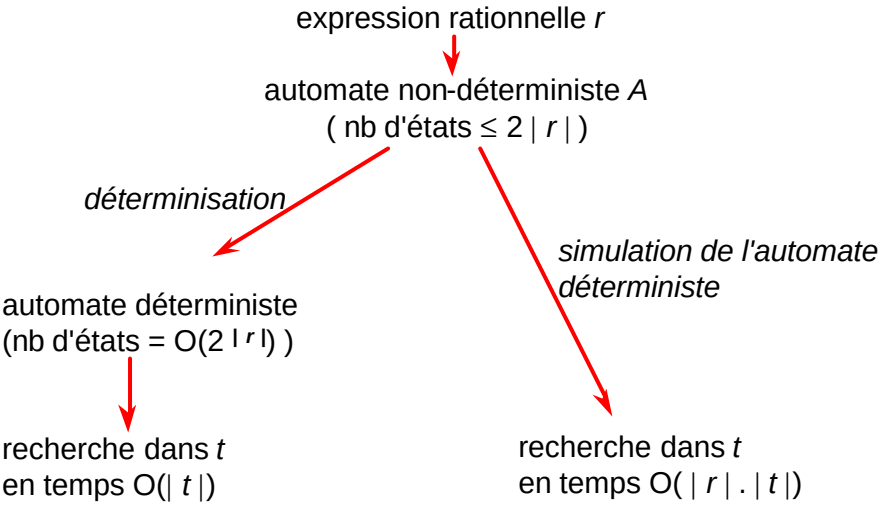
UMLV ©

```
booléen recherche(automate A, mot t){
     ensemble E;
     E ¬ clôture(A.initial);
     tant que(non fin de t){
          car ¬ lettre suivante de t;
          E ¬ clôture({A.initial}È transition(E,car));
          si(A.terminal Î E) retour(vrai);
     }
     retour(faux);
}
```

Lecture séquentielle du texte.
Mémorisation (tampon) d'une seule lettre de *t*.
Temps : O(nb états de *A* . | *t* |)

(si bonne gestion de l 'ensemble *E*)

Compromis temps-espace

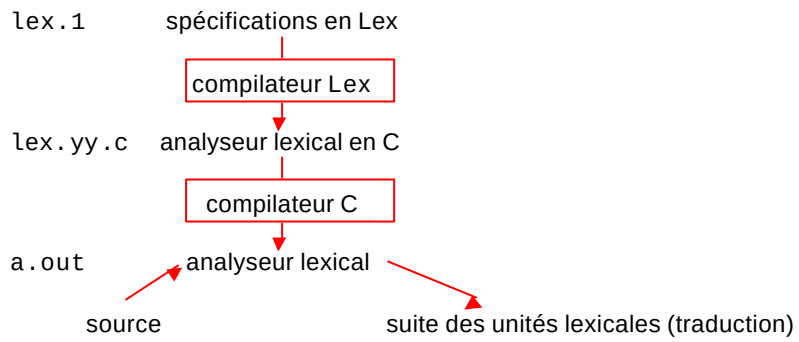


	déterminisation	*simulation*
temps de la recherche	O(| *t* |)	O(| *r* | . | *t* |)
espace	O(2^| *r* |)	O(| *r* |)

UMLV ©

LEX

Création d'analyseurs lexicaux ou de traducteurs



Spécifications : déclarations
%%
règles de traductions
%%
procédures auxiliaires

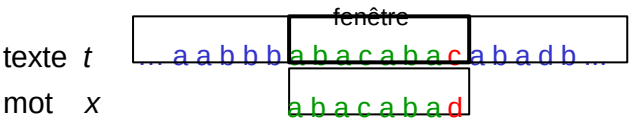
UMLV ©

```
% {/* définitions des constantes littérales */
         PPQ, PPE, EGA, DIF, PGQ, PGE,
         SI, ALORS, SINON, ID, NB, OPREL
% }
/* définitions régulières */
délim       [ \t\n]
bl          {delim}+
lettre      [A-Za-z]
chiffre     [0-9]
id          {lettre}+({lettre}|{chiffre})*
nombre      {chiffre}+(\.{chiffre}+)?(E[+\-]?{chiffre}+)?
%%
{bl}        {/* pas d'action et pas de retour */ }
si          {return (SI) ; }
alors       {return (ALORS) ; }
sinon       {return (SINON) ; }
{id}        {yylval = RangerId ( ) ; return (ID) ; }
{nombre}    {yylval = RangerNb ( ) ; return (NB) ; }
"<"         {yylval = PPQ ; return (OPREL) ; }
"<="        {yyval = PPE ; return (OPREL) ; }
"="         {yyval = EGA ; return (OPREL) ; }
"<>"        {yyval = DIF ; return (OPREL) ; }
">"         {yyval = PGQ ; return (OPREL) ; }
">="        {yyval = PGE ; return (OPREL) ; }
```

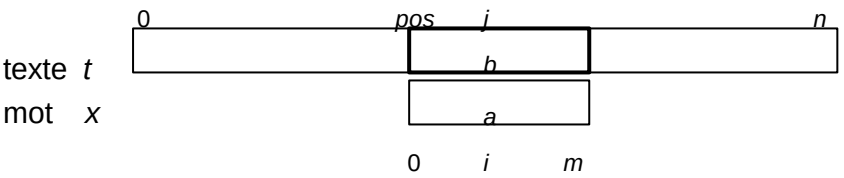
```
%%
RangerId(){
      /* procédure qui range dans la table des symboles
      le lexème dont le premier caractère est pointé
      par yytext et dont la longueur est yyleng,
      et retourne un pointeur sur son entrée */
...
}

RangerNb(){
      /* procédure similaire pour ranger un lexème
      qui est un nombre */
...
}
```


Recherche d'un mot

Recherche des positions du mot *x* dans le texte *t*
Par balayages (gauche-droite) et décalages



Variables de travail



Algorithme naïf

UMLV ©

```
Naif(mot x, longueur m, mot t, longueur n){
      j ¬ 0;  i ¬ 0;
      tant que (i < m) et (j < n){
            si (x[i] = t[j]){
                  j ¬ j+1;  i ¬ i+1;
            } sinon {
                  j ¬ j-i+1;  i ¬ 0;
            fin ;
      si (i > m) écrire(mot trouvé à la position j-i);
      sinon écrire(pas d'occurrence de x dans t);
}
```

Temps maximal O(*m.n*)

» *m.n* comparaisons au pire

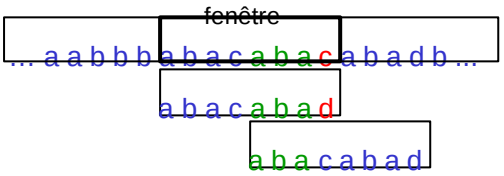
Temps moyen O(*n*)

» 2.*n* comparaisons moyennes sur alphabet binaire (équiprobabilité et indépendance)

Méthode de Knuth, Morris & Pratt

UMLV ©

KMP : balayages gauche-droite et décalages compatibles

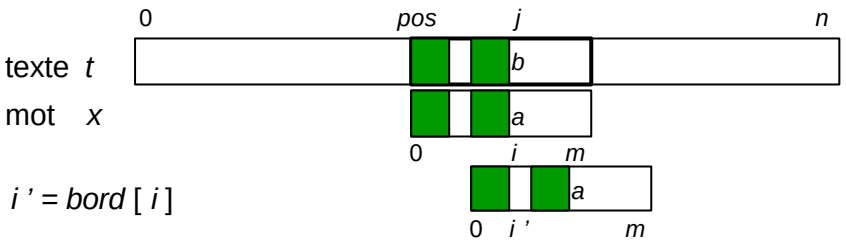


Bords : Bord(abacaba) = aba

Longueur des bords des préfixes : *bord* [*i*] = |Bord(*x*[0..*i*-1])|

i	0	1	2	3	4	5	6	7	8
x[*i*]	a	b	a	c	a	b	a	d	
bord [*i*]	-1	0	0	1	0	1	2	3	0

Recherche KMP



Algorithme


```
KMP(mot x, longueur m, mot t, longueur n){
      j ¬ 0 ;  i ¬ 0 ;
      tant que (i < m) et (j < n){
            tant que (i > 0) et (x[i] ≠ t[j])
                  i ¬ bord[i] ;
            j ¬ j+1 ;   i ¬ i+1 ;
      }
      si (i = m) écrire(mot trouvé à la position j-m);
      sinon écrire(pas d'occurrence de x dans t);
}
```

Temps O(*n*)

moins de 2.*n* comparaisons

recherche séquentielle (*j* ne décroît jamais)

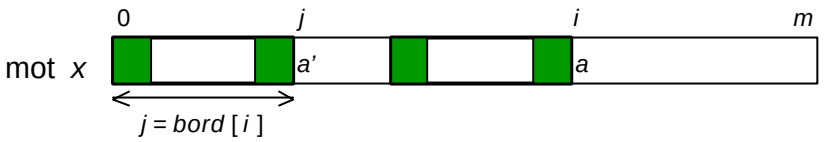
recherche linéaire mais pas en temps réel

Calcul des bords

UMLV ©

Bords : Bord(*u*) = plus long préfixe et suffixe propre de *u*
Note : le bord d 'un bord est un bord

Bord(a b a c a b a **c**) = a b a c
Bord(a b a c a b a **b**) = a b
Bord(a b a c a b a **d**) = ε



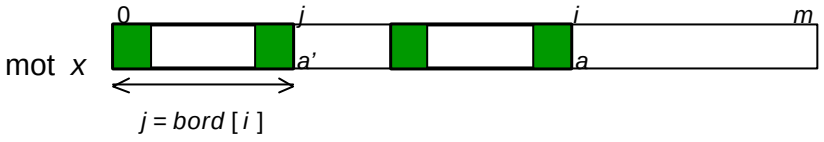
si *a*=*a'* *bord* [*i*+1] = *bord* [*i*] +1 = *j* +1
sinon faire le calcul avec *a* sur Bord(*x*[0 .. *j*-1])

Algorithme

UMLV ©

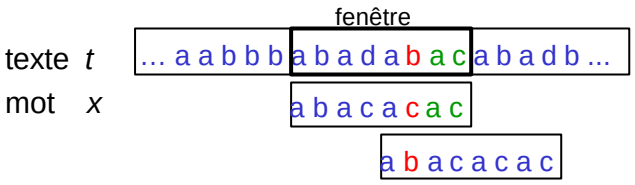
```
BORDS(mot x, longueur m){
      bord[0] ¬ -1 ;
      pour i ¬ 0 à m-1 faire {
            j ¬ bord[i] ;
            tant que (j ³ 0) et (x[i] ¹ x[j])
                  j ¬ bord[j] ;
            bord[i+1] ¬ j+1 ;
      }
      retour bord ;
}
```

Temps O(*m*), moins de 2.*m* comparaisons}
Comme la recherche !



Méthode de Boyer-Moore (simplifiée)

BM : balayages droite-gauche et décalages compatibles avec la lettre différente du texte

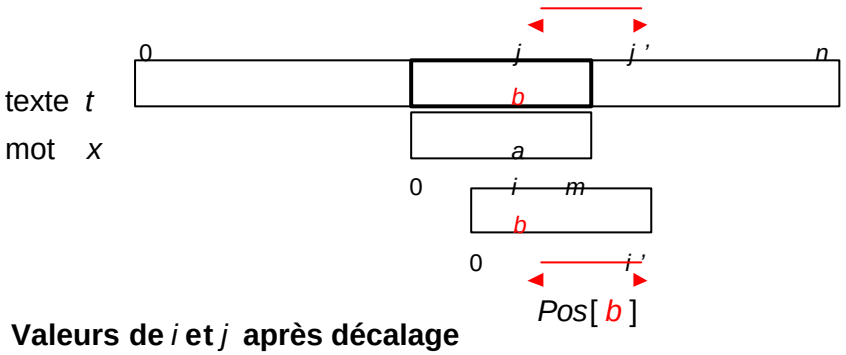


Positions

a	a	b	c	d	e	...
Pos_x[*a*]	1	6	2	8	8	...

Pos_x[b]=6

Recherche BM



Valeurs de *i* et *j* après décalage

i ' = *m*

j ' = *j* + Pos_x[*t* [*j*]] **si** Pos_x[*t* [*j*]] > *m* - *i* + 1

m - *i* + 1 **sinon**

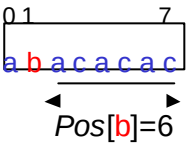
Algorithme BM


```
BM(mot x, longueur m, mot t, longueur n){
      j ¬ m-1 ;  j ¬ m-1 ;
      tant que (i ³ 0) et (j < n)
            si(x[i] = t[j]){
                  j ¬ j-1 ;   i ¬ i-1 ;
            } sinon {
                  j ¬ j + max{Pos[j],m-i+1} ;   i ¬ m-1 ;
            }
      si (i = -1) écrire(mot trouvé à la position j+1);
      sinon écrire(pas d'occurrence de x dans t);
}
```

Temps maximal O(*m.n*)
Temps, sur des textes en langue naturelle, proche du minimal : Θ(*n* / *m*)

Calcul de *Pos*


```
Positions(mot x, longueur m){
      pour chaque lettre a faire
            Pos[a] ¬ m ;
      pour i ¬ 0 à m-2 faire
            Pos[x[i]] ¬ m-i-1 ;
      retour Pos ;
}
```



Temps O(*m* + card *A*)