Proiectarea algoritmilor: Căutare peste șiruri

Ștefan Ciobâcă, Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

PA 2015/2016

1. Algoritmul Knuth-Morris-Pratt

2. Expresii regulate

Plan

Algoritmul naiv[1]

b	*a*	*c*	*b*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*c*	*b*	*a*	*b*
				a	*b*	*a*	*b*	*a*	*c*	*a*				

[1] Exemplu din [CLRS]

Algoritmul naiv[1]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$										
				a	b	a	b	a	c	a				

[1]Exemplu din [CLRS]

Algoritmul naiv[1]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$	$=$									
				a	b	a	b	a	c	a				

[1] Exemplu din [CLRS]

Algoritmul naiv[1]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$	$=$	$=$								
				a	b	a	b	a	c	a				

[1] Exemplu din [CLRS]

Algoritmul naiv[1]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$	$=$	$=$	$=$							
				a	b	a	b	a	c	a				

[1] Exemplu din [CLRS]

Algoritmul naiv[1]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$	$=$	$=$	$=$	$=$						
				a	b	a	b	a	c	a				

[1] Exemplu din [CLRS]

Algoritmul naiv[1]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$	$=$	$=$	$=$	$=$	$\neq$					
				a	b	a	b	a	c	a				

[1]Exemplu din [CLRS]

Intuiția[2]

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b
				$=$	$=$	$=$	$=$	$=$	$\neq$					
				a	b	a	b	a	c	a				
					a	b	a	b	a	c	a			
						a	b	a	b	a	c	a		

[2] Exemplu din [CLRS]

Intuiția[3]

?	?	?	?	a	b	a	b	a	?	?	?	?	?	?
				$=$	$=$	$=$	$=$	$=$	$\neq$					
				a	b	a	b	a	?	?				
					a	b	a	b	?	?	?			
						a	b	a	?	?	?	?		

[3] Exemplu din [CLRS]

Intuiția[3]

?	?	?	?	*a*	*b*	*a*	*b*	*a*	?	?	?	?	?	?
				$=$	$=$	$=$	$=$	$=$	$\neq$					
				a	*b*	*a*	*b*	*a*	?	?				
					a	*b*	*a*	*b*	?	?	?			
						a	*b*	*a*	?	?	?	?		

Pentru pattern-ul *ababaca*, dacă la o poziție i se potrivesc exact 5 caractere, nu există nicio șansă ca pattern-ul să se potrivească la poziția $i + 1$.

[3] Exemplu din [CLRS]

Ideea

?	?	?	?	x_1	x_2	x_3	x_4	x_5	x_6	x_7	?	?	?	?	?	?	?	?
				$=$	$=$	$=$	$=$	$=$	$=$	$=$	$\neq$							
				x_1	x_2	x_3	x_4	x_5	x_6	x_7	?							
								$=$	$=$	$=$								
								x_1	x_2	x_3	x_4	x_5	x_6	x_7	?			

Ideea

?	?	?	?	x_1	x_2	x_3	x_4	x_1	x_2	x_3	?	?	?	?	?	?	?	?
				$=$	$=$	$=$	$=$	$=$	$=$	$=$	$\neq$							
				x_1	x_2	x_3	x_4	x_1	x_2	x_3	?							
								$=$	$=$	$=$								
								x_1	x_2	x_3	x_4	x_1	x_2	x_3	?			

Ideea

?	?	?	?	x_1	$\ldots$	x_k	$\ldots$	x_1	$\ldots$	x_k	?	?	?	?
				$=$	$=$	$=$	$=$	$=$	$=$	$=$	$\neq$			
				x_1	$\ldots$	x_k	$\ldots$	x_1	$\ldots$	x_k	?			

Ideea

?	?	?	?	x_1	$\dots$	x_k	$\dots$	x_1	$\dots$	x_k	?	?	?	?
				$=$	$=$	$=$	$=$	$=$	$=$	$=$	$\neq$			
				x_1	$\dots$	x_k	$\dots$	x_1	$\dots$	x_k	?			

Ne interesează cea mai mare valoarea a lui k astfel încât $x_1 \dots x_k$ să fie atât prefix cât și sufix al părții din pattern care s-a potrivit.

Notații

1. $s \sqsubset t$ dacă s este prefix al lui t

Notații

1. $s \sqsubset t$ dacă s este prefix al lui t
2. $s \sqsupset t$ dacă s este sufix al lui t

Notații

1. $s \sqsubset t$ dacă s este prefix al lui t
2. $s \sqsupset t$ dacă s este sufix al lui t
3. Exemple: $\epsilon \sqsubset aba$

Notații

1. $s \sqsubset t$ dacă s este prefix al lui t
2. $s \sqsupset t$ dacă s este sufix al lui t
3. Exemple: $\epsilon \sqsubset aba$, $aba \sqsubset ababa$

Notații

1. $s \sqsubset t$ dacă s este prefix al lui t
2. $s \sqsupset t$ dacă s este sufix al lui t
3. Exemple: $\epsilon \sqsubset aba$, $aba \sqsubset ababa$, $aba \sqsubset aba$

Notații

1. $s \sqsubset t$ dacă s este prefix al lui t
2. $s \sqsupset t$ dacă s este sufix al lui t
3. Exemple: $\epsilon \sqsubset aba$, $aba \sqsubset ababa$, $aba \sqsubset aba$, $aa \sqsupset abaa$.

Funcția eșec

1. $t[0..m-1]$ (patternul)

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1						

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0					

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0	0				

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0	0	1			

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0	0	1	2		

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0	0	1	2	3	

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0	0	1	2	3	0

Funcția eșec

1. $t[0..m-1]$ (patternul)
2. $s[0..n-1]$ (textul (subiectul))
3. Funcția eșec $f : \{0, \ldots, m-1\} \to \mathbb{N}$
4. $f(i) =$ cel mai lung prefix propriu al $t[0..i-1]$ care este și sufix al $t[0..i-1]$)
5. Exemplu:

i	0	1	2	3	4	5	6
$t[i]$	a	b	a	b	a	c	a
$f(i)$	-1	0	0	1	2	3	0

Exemplu pe tablă.

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
    1. `if k == m` (s-au potrivit toate cele m caractere)

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
    1. `if k == m` (s-au potrivit toate cele m caractere)
        - `return i;`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
    1. `if k == m` (s-au potrivit toate cele m caractere)
        - `return i;`
    2. `else`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
   1. `if k == m` (s-au potrivit toate cele m caractere)
      - `return i;`
   2. `else`
      1. `if s[i + k] == t[k]` (încă un caracter se potrivește)

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
   1. `if k == m` (s-au potrivit toate cele m caractere)
      - `return i;`
   2. `else`
      1. `if s[i + k] == t[k]` (încă un caracter se potrivește)
         `k = k + 1;`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
    1. `if k == m` (s-au potrivit toate cele m caractere)
        - `return i;`
    2. `else`
        1. `if s[i + k] == t[k]` (încă un caracter se potrivește)
           `k = k + 1;`
        2. `else` (nepotrivire)

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
   1. `if k == m` (s-au potrivit toate cele m caractere)
      - `return i;`
   2. `else`
      1. `if s[i + k] == t[k]` (încă un caracter se potrivește)
         `k = k + 1;`
      2. `else` (nepotrivire)
         `i = i + k - f[k];`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
   1. `if k == m` (s-au potrivit toate cele m caractere)
      - `return i;`
   2. `else`
      1. `if s[i + k] == t[k]` (încă un caracter se potrivește)
         `k = k + 1;`
      2. `else` (nepotrivire)
         `i = i + k - f[k]; k = max(f[k], 0);`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
   1. `if k == m` (s-au potrivit toate cele m caractere)
      - `return i;`
   2. `else`
      1. `if s[i + k] == t[k]` (încă un caracter se potrivește)
         `k = k + 1;`
      2. `else` (nepotrivire)
         `i = i + k - f[k]; k = max(f[k], 0);`
5. `return -1;`

Algoritmul KMP

1. `compute f[0..m-1];` (vom vedea mai târziu cum)
2. `i = 0;` (încerc să găsesc o potrivire pe poziția $i = 0$)
3. `k = 0;` (pentru moment sunt $k = 0$ poziții care se potrivesc)
4. `while (i + m <= n)`
   1. `if k == m` (s-au potrivit toate cele m caractere)
      - `return i;`
   2. `else`
      1. `if s[i + k] == t[k]` (încă un caracter se potrivește)
         `k = k + 1;`
      2. `else` (nepotrivire)
         `i = i + k - f[k]; k = max(f[k], 0);`
5. `return -1;`

Timpul de execuție

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

Timpul de execuție

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

1. Observație: pentru orice k, $f[k] < k$.

Timpul de execuție

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

1. Observație: pentru orice k, $f[k] < k$.
2. Observație la fiecare iterație a buclei `while`, valoarea expresiei $2i + k$ crește cu cel puțin o unitate (vezi pe tablă).

Timpul de execuție

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

1. Observație: pentru orice k, $f[k] < k$.
2. Observație la fiecare iterație a buclei `while`, valoarea expresiei $2i + k$ crește cu cel puțin o unitate (vezi pe tablă).
3. Observație: ce valoarea poate avea $2i + k$ (maxim)?

Timpul de execuție

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

1. Observație: pentru orice k, $f[k] < k$.
2. Observație la fiecare iterație a buclei `while`, valoarea expresiei $2i + k$ crește cu cel puțin o unitate (vezi pe tablă).
3. Observație: ce valoarea poate avea $2i + k$ (maxim)?
4. Concluzie: câte iterații ale buclei `while` sunt executate?

Algoritmul KMP (prezentare alternativă)

```
KMP(s, n, t, m, f) {
  i = 0;
  k = 0;
  while (i < n) {
    while (k != -1) && (p[k] != s[i])
      k = f[k];
    if (k = m-1)
      return i-m+1; /* gasit p in s */
    else {
      i = i+1;
      k = k+1;
    }
  }
  return -1; /* p nu apare in s */
}
```

Algoritmul KMP (prezentare alternativă)

```
KMP(s, n, t, m, f) {
  i = 0;
  k = 0;
  while (i < n) {
    while (k != -1) && (p[k] != s[i])
      k = f[k];
    if (k = m-1)
      return i-m+1; /* gasit p in s */
    else {
      i = i+1;
      k = k+1;
    }
  }
  return -1; /* p nu apare in s */
}
```

Corelație: variabila i din acest program "ține minte" valoarea $i + k$ din programul precedent.

Calculul funcției eșec

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

Calculul funcției eșec

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

O implementare naivă poate avea complexitatea $O(m^3)$ (exercițiu pentru acasă).

Calculul funcției eșec

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

O implementare naivă poate avea complexitatea $O(m^3)$ (exercițiu pentru acasă).
Dacă presupunem că $f[0..i-1]$ a fost deja calculat, cum calculăm eficient $f[i]$?

Calculul funcției eșec

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

O implementare naivă poate avea complexitatea $O(m^3)$ (exercițiu pentru acasă).
Dacă presupunem că $f[0..i-1]$ a fost deja calculat, cum calculăm eficient $f[i]$?

1. Dacă $t[i-1] = t[f[i-1]]$, atunci $f[i] = f[i-1] + 1$.

Calculul funcției eșec

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

O implementare naivă poate avea complexitatea $O(m^3)$ (exercițiu pentru acasă).

Dacă presupunem că $f[0..i-1]$ a fost deja calculat, cum calculăm eficient $f[i]$?

1. Dacă $t[i-1] = t[f[i-1]]$, atunci $f[i] = f[i-1] + 1$.
2. (altfel,) dacă $t[i-1] = t[f[f[i-1]]]$, atunci $f[i] = f[f[i-1]] + 1$.

Calculul funcției eșec

a	b	a	b	a	c	a
-1	0	0	1	2	3	0

O implementare naivă poate avea complexitatea $O(m^3)$ (exercițiu pentru acasă).

Dacă presupunem că $f[0..i-1]$ a fost deja calculat, cum calculăm eficient $f[i]$?

1. Dacă $t[i-1] = t[f[i-1]]$, atunci $f[i] = f[i-1] + 1$.
2. (altfel,) dacă $t[i-1] = t[f[f[i-1]]]$, atunci $f[i] = f[f[i-1]] + 1$.
3. etc.

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`
3. `for (i = 1; i < m; ++i)` (calculez $f[i]$)

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`
3. `for (i = 1; i < m; ++i)` (calculez $f[i]$)
   (știu că $t[0..i-2]$ are un prefix propriu de lungime k care este și sufix)

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`
3. `for (i = 1; i < m; ++i)` (calculez $f[i]$)
   (știu că $t[0..i-2]$ are un prefix propriu de lungime k care este și sufix)
   1. `while k != -1 && t[k] != t[i - 1]`

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`
3. `for (i = 1; i < m; ++i)` (calculez $f[i]$)
   (știu că $t[0..i-2]$ are un prefix propriu de lungime k care este și sufix)
   1. ```
      while k != -1 && t[k] != t[i - 1]
            k = f[k]
      ```
      (iau un prefix mai mic)
      ```

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`
3. `for (i = 1; i < m; ++i)` (calculez $f[i]$)
   (știu că $t[0..i-2]$ are un prefix propriu de lungime k care este și sufix)
   1. `while k != -1 && t[k] != t[i - 1]`
      `k = f[k]` (iau un prefix mai mic)
   2. `k = k + 1;` (adaug caracterul $t[k] = t[i-1]$)

Calculul funcției eșec

1. `f[0] = -1;` (deoarece $t[0..-1] = \epsilon$ nu are niciun prefix propriu)
2. `k = -1;`
3. `for (i = 1; i < m; ++i)` (calculez $f[i]$)
   (știu că $t[0..i-2]$ are un prefix propriu de lungime k care este și sufix)
   1. `while k != -1 && t[k] != t[i - 1]`
      `k = f[k]` (iau un prefix mai mic)
   2. `k = k + 1;` (adaug caracterul $t[k] = t[i-1]$)
   3. `f[i] = k;`

Plan

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută.

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*.

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din* Σ *este o expresie regulată;*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul Σ este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din Σ este o expresie regulată;*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din* Σ *este o expresie regulată;*
- *dacă* e_1, e_2 *sunt expresii regulate, atunci* $e_1 e_2$ *și* $e_1 + e_2$ *sunt expresii regulate;*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din* Σ *este o expresie regulată;*
- *dacă* e_1, e_2 *sunt expresii regulate, atunci* $e_1 e_2$ *și* $e_1 + e_2$ *sunt expresii regulate;*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din* Σ *este o expresie regulată;*
- *dacă* e_1, e_2 *sunt expresii regulate, atunci* $e_1 e_2$ *și* $e_1 + e_2$ *sunt expresii regulate;*
- *dacă* e *este expresie regulată, atunci* (e) *și* e^* *sunt expresii regulate.*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul Σ este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din Σ este o expresie regulată;*
- *dacă e_1, e_2 sunt expresii regulate, atunci $e_1 e_2$ și $e_1 + e_2$ sunt expresii regulate;*
- *dacă e este expresie regulată, atunci (e) și e^* sunt expresii regulate.*

Definiție

În această secțiune considerăm cazul când "pattern"-ul constituie doar o specificație a ceea ce se caută în sensul că el desemnează o mulțime de șiruri pentru care se caută. Numim o astfel de specificație *"pattern" generalizat*. Un alt mod de a specifica "pattern"-uri generalizate îl constituie expresiile regulate.

Definiție

Mulțimea expresiilor regulate *peste alfabetul* Σ *este definită recursiv astfel:*

- ε, *empty sunt expresii regulate*
- *orice caracter din* Σ *este o expresie regulată;*
- *dacă* e_1, e_2 *sunt expresii regulate, atunci* $e_1 e_2$ *și* $e_1 + e_2$ *sunt expresii regulate;*
- *dacă* e *este expresie regulată, atunci* (e) *și* e^* *sunt expresii regulate.*

Arborele sintactic abstract: pe tabla.

Legătura cu pachetul <regex> din C++

`<regex>`	expresia regulata
`[abc]`	$a + b + c$
`\d` sau `[[:digit:]]`	$0 + 1 + \cdots + 9$
`[[:digit:]]*`	$(0 + 1 + \cdots + 9)^*$
`[[:digit:]]+`	$(0 + 1 + \cdots + 9)(0 + 1 + \cdots + 9)^*$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ *definit de o expresie regulată* e *este definit recursiv astfel:*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) =$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) =$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathit{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathit{empty}) = \emptyset$
- *dacă e este un caracter atunci* $L(e) = \{e\}$*;*
- *dacă* $e = e_1 e_2$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*
- *dacă $e = e_1 e_2$ atunci $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci* $L(e) = \{e\}$*;*
- *dacă* $e = e_1 e_2$ *atunci* $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$*;*
- *dacă* $e = e_1 + e_2$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*
- *dacă $e = e_1 e_2$ atunci $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;*
- *dacă $e = e_1 + e_2$ atunci $L(e) = L(e_1) \cup L(e_2)$;*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci* $L(e) = \{e\}$;
- *dacă* $e = e_1 e_2$ *atunci* $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;
- *dacă* $e = e_1 + e_2$ *atunci* $L(e) = L(e_1) \cup L(e_2)$;
- *dacă* $e = e_1^*$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*
- *dacă $e = e_1 e_2$ atunci $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;*
- *dacă $e = e_1 + e_2$ atunci $L(e) = L(e_1) \cup L(e_2)$;*
- *dacă $e = e_1^*$ atunci $L(e) = \cup_k L(e_1^k)$, unde* $L(e_1^0) = \{\varepsilon\}, L(e_1^{k+1}) = L(e_1^k)L(e_1)$;

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*
- *dacă $e = e_1 e_2$ atunci $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;*
- *dacă $e = e_1 + e_2$ atunci $L(e) = L(e_1) \cup L(e_2)$;*
- *dacă $e = e_1^*$ atunci $L(e) = \cup_k L(e_1^k)$, unde* $L(e_1^0) = \{\varepsilon\}, L(e_1^{k+1}) = L(e_1^k)L(e_1)$;
- *dacă $e = (e_1)$*

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*
- *dacă $e = e_1 e_2$ atunci $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;*
- *dacă $e = e_1 + e_2$ atunci $L(e) = L(e_1) \cup L(e_2)$;*
- *dacă $e = e_1^*$ atunci $L(e) = \cup_k L(e_1^k)$, unde* $L(e_1^0) = \{\varepsilon\}, L(e_1^{k+1}) = L(e_1^k)L(e_1)$;
- *dacă $e = (e_1)$ atunci $L(e) = L(e_1)$.*

Exemplu: Fie alfabetul $A = \{a, b, c\}$. Avem $L(a(b + a)c) =$

Limbajul definit de o expresie regulată

Definiție

Mulțimea de șiruri (limbajul) $L(e)$ definit de o expresie regulată e este definit recursiv astfel:

- $L(\varepsilon) = \{\varepsilon\}$ *(ε este șirul vid (de lungime zero)),* $L(\mathtt{empty}) = \emptyset$
- *dacă e este un caracter atunci $L(e) = \{e\}$;*
- *dacă $e = e_1 e_2$ atunci $L(e) = L(e_1)L(e_2) = \{w_1 w_2 \mid w_1 \in L(e_1), w_2 \in L(e_2)\}$;*
- *dacă $e = e_1 + e_2$ atunci $L(e) = L(e_1) \cup L(e_2)$;*
- *dacă $e = e_1^*$ atunci $L(e) = \cup_k L(e_1^k)$, unde $L(e_1^0) = \{\varepsilon\}, L(e_1^{k+1}) = L(e_1^k)L(e_1)$;*
- *dacă $e = (e_1)$ atunci $L(e) = L(e_1)$.*

Exemplu: Fie alfabetul $A = \{a, b, c\}$. Avem $L(a(b + a)c) = \{abc, aac\}$ și $L((ab)^*) = \{\varepsilon, ab, abab, ababab, \ldots\} = \{(ab)^k \mid k \geq 0\}$. sfex

Automatul asociat unei expresii regulate

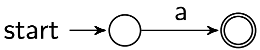
– cazul de baza

e este o litera (un simbol) $a \in \Sigma$

Automatul asociat unei expresii regulate

– cazul de baza

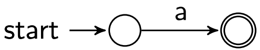
e este o litera (un simbol) $a \in \Sigma$



Automatul asociat unei expresii regulate

– cazul de baza

e este o litera (un simbol) $a \in \Sigma$

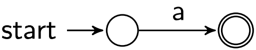


e este ε

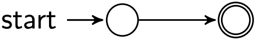
Automatul asociat unei expresii regulate

– cazul de baza

e este o litera (un simbol) $a \in \Sigma$



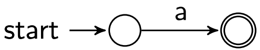
e este ε



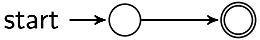
Automatul asociat unei expresii regulate

– cazul de baza

e este o litera (un simbol) $a \in \Sigma$



e este ε

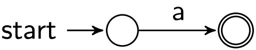


e este *empty*

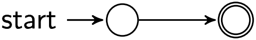
Automatul asociat unei expresii regulate

– cazul de baza

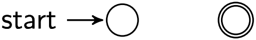
e este o litera (un simbol) $a \in \Sigma$



e este ε



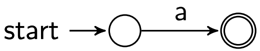
e este *empty*



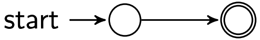
Automatul asociat unei expresii regulate

– cazul de baza

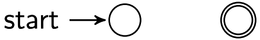
e este o litera (un simbol) $a \in \Sigma$



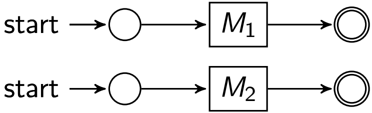
e este ε



e este *empty*



pentru cazul inductiv presupunem:



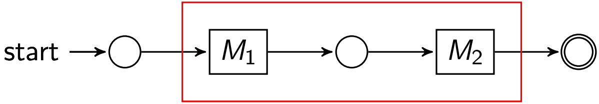
Automatul asociat unei expresii regulate

Automatul asociat unei expresii regulate

$e = e_1 e_2$:

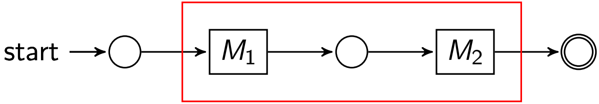
Automatul asociat unei expresii regulate

$e = e_1 e_2$:



Automatul asociat unei expresii regulate

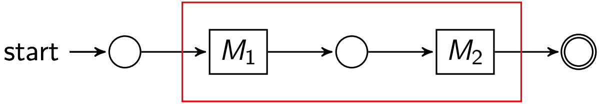
$e = e_1 e_2$:



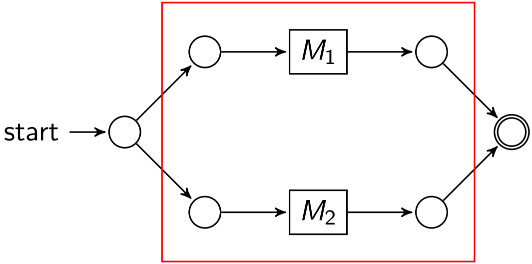
$e = e_1 + e_2$:

Automatul asociat unei expresii regulate

$e = e_1 e_2$:



$e = e_1 + e_2$:

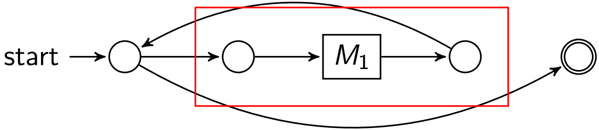


Automatul asociat unei expresii regulate

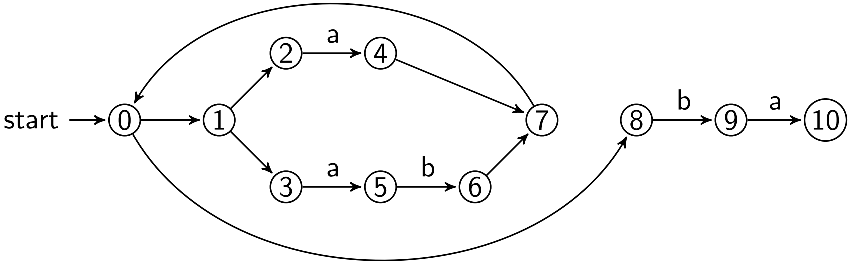
$e = e_1^*$:

Automatul asociat unei expresii regulate

$e = e_1^*$:



Exemplu



Detaliile procesului de construție pe tablă

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite:

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$,

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări,

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul,

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile,

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială,

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală
- dacă $M(e)$ este automatul asociat lui e, atunci $L(M(e)) = L(e)$

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală
- dacă $M(e)$ este automatul asociat lui e, atunci $L(M(e)) = L(e)$
- tranzițiile neetichetate se numesc și ε-tranziții (sau spontane)

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală
- dacă $M(e)$ este automatul asociat lui e, atunci $L(M(e)) = L(e)$
- tranzițiile neetichetate se numesc și ε-tranziții (sau spontane)
- automatul construit direct din definiție este în general nedeterminist (și neminimal)

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală
- dacă $M(e)$ este automatul asociat lui e, atunci $L(M(e)) = L(e)$
- tranzițiile neetichetate se numesc și ε-tranziții (sau spontane)
- automatul construit direct din definiție este în general nedeterminist (și neminimal)
- costisitor de aplicat în practică

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală
- dacă $M(e)$ este automatul asociat lui e, atunci $L(M(e)) = L(e)$
- tranzițiile neetichetate se numesc și ε-tranziții (sau spontane)
- automatul construit direct din definiție este în general nedeterminist (și neminimal)
- costisitor de aplicat în practică
- se poate construi un automat echivalent determinist?

Automate nedeterministe

- Automatele asociate expresiilor regulate sunt cazuri particulare de automate finite: $M = (Q, \Sigma, \delta, q_0, Q_f)$, unde Q este mulțimea de stări, Σ alfabetul, $\delta : Q \times A \to \mathcal{Q}$ tranzițiile, $q_0 \in Q$ starea inițială, $Q_f \subseteq Q$ stările finale
- limbajul acceptat $L(M)$ este mulțimea de cuvinre ce descriu parcursuri de la starea inițială la o stare finală
- dacă $M(e)$ este automatul asociat lui e, atunci $L(M(e)) = L(e)$
- tranzițiile neetichetate se numesc și ε-tranziții (sau spontane)
- automatul construit direct din definiție este în general nedeterminist (și neminimal)
- costisitor de aplicat în practică
- se poate construi un automat echivalent determinist?
- răspunsul este afirmativ (automatele finite nedeterministe au aceeași pute de acceptare ca și cele deterministe), dar cu anumite costuri (a se vedea slide-urilr următoare)

Construcția unui automat determinist echivalent

Fie N un automat nedeterminist cu mulțimea de stări Q. Construim un automat determinsit D astfel:

- mulțimea de stări este $\mathcal{P}(Q)$ (număr exponențial de stări!!!)
- există tranziție etichetată cu a de la Q_1 la Q_2 dacă și numai dacă Q_2 este mulțimea tuturor stărilor q_2 cu proprietatea că există $q_1 \in Q_1$ și tranziție etichetată cu a de la q_1 la q_2 în N
- starea inițială a lui D este $\{q_0\}$, unde q_0 este starea inițială a lui N
- o submulțime Q_f este stare finală dacă și numai dacă include o stare finală q_f a lui N

Exemplu pe tablă.

Construcția de mai sus se poate îmbunătăți utilizând un algoritm bazat pe derivativele Brzozowski.

Derivativele Brzozowski

Derivativele unei expresii regulate (Brzozowski, 1964):

$$\delta_a(empty) = empty \qquad \varepsilon?(empty) = empty$$

$$\delta_a(\varepsilon) = empty \qquad \varepsilon?(\varepsilon) = \varepsilon$$

$$\delta_a(b) = \begin{cases} \varepsilon & , b = a \\ empty & , b \neq a \end{cases} \qquad \varepsilon?(b) = empty$$

$$\delta_a(e_1 e_2) = \delta_a(e1)e_2 + \varepsilon?(e1)\delta_a(e_2) \qquad \varepsilon?(e_1 e_2) = \varepsilon?(e_1)\varepsilon?(e_2)$$

$$\delta_a(e_1 + e_2) = \delta_a(e_1) + \delta_a(e_2) \qquad \varepsilon?(e_1 + e_2) = \varepsilon?(e_1) + \varepsilon?(e_2)$$

$$\delta_a(e^*) = \delta_a(e)e^* \qquad \varepsilon?(e^*) = \varepsilon$$

Extensia la cuvinte: $\delta_\varepsilon(e) = e$, $\delta_{wa}(e) = \delta_a(\delta_w(e))$

Simplificări

concatenarea și $+$ sunt asociative, $+$ este și comutativă

$e + e = e$

$e + empty = empty + e = e$

$e\ empty = empty\ e = empty$

$e\varepsilon = \varepsilon e = e$

Proprietatea fundamentală

Theorem (Brzozowski)

Mulțimea derivatelor unei expresii $\{\delta_w(e) \mid w \in A^*\}$ *este finită.*

Exemplu:

$\{\delta_w(\texttt{(ab + a)}^* \texttt{ b a}) \mid w \in A^*\} =$

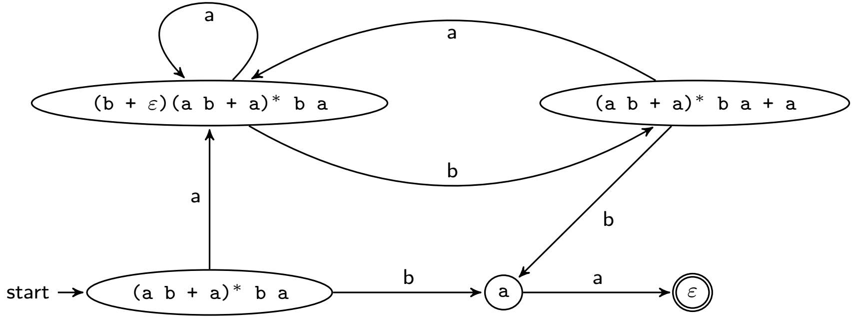
$\{\texttt{(b + }\varepsilon\texttt{)((a b + a)}^* \texttt{ b a)}, \texttt{(a b + a)}^* \texttt{ b a}, \texttt{a}, \varepsilon, \mathit{empty}\}$

Construcția automatului (Brzozowski)

- mulțimea de stări este mulțimea derivatelor
- există tranziție etichetată cu a de la q_1 la q_2 dacă și numai dacă q_1 corespunde unei derivate $\delta_w(e)$ și q_2 corespunde derivatei $\delta_{wa}(e)$ pentru un $w \in A^*$;
- starea inițială este $e = \delta_\varepsilon(e)$
- o stare q este finală (de acceptare) dacă și numai dacă corespunde unei derivate $\delta_w(e)$ și $\varepsilon?(\delta_w(e)) = \varepsilon$.

Exemplu

$e = $ `(a b + a)* b a`



Construcția unui automat determinist

(Berry, Setti: From Regular Expressions To Deterministic Automata, 1986)

O continuare a lui a în e este orice expresie $\delta_{wa}(e) \neq empty$.

- se marchează simbolurile din e ca fiind distincte; fie e' expresia obținută (de exemplu $e = (ab + b)^* ba$ este transformată în $e' = (a_1 b_2 + b_3)^* b_4 a_5$)
- se construiește automatul M' pentru e' urmând ideea din algoritmul lui Brzozowski:
  - M' are o stare pentru fiecare continuare a unui simbol marcat în e'
  - există tranziție de la q_1 la q_2 dacă și numai dacă q_1 corespunde unei continuări C, C, C poate genera un cuvânt care începe cu a ($\delta_{wa}(e) \neq empty$) și q_2 corespunde continuării lui a
  - starea inițială este e'
  - q este o stare finală (de acceptare) dacă și numai dacă ea corespunde unei continuări C și $\varepsilon?(C) = \varepsilon$
- se elimină mărcile din M'
- se determinizează M' construind M ale cărui stări sunt submulțimi de stări ale lui M'

Construcții mai performante

- utilizând funcțiile `first` și `follow` (Berry, Setti, 1986)
- paralelizare (Myer, A Four Russians Algorithm for Regular Expression Pattern Matching)
- o altă construcție pentru automatul nedeterminist este Glushkov-McNaughton-Yamada (1960-1961), care poate fi si paralelizată (Navarro & Raffinot, 2004)

Complexitatea căutării cu expresii regulate

Presupunem că lungimea expresiei regulate este m (numărul de caractere fără operatori) și $m_\Sigma = |\Sigma \cup \{\cdot, +, *\}|$.

Theorem (Thomson, 1968)

Problema căutării cu expresii regulate poate fi rezolvată în timpul $O(mn)$ cu automate nedeterministe și spațiu $O(m)$.

Theorem (Kleene, 1956)

Problema căutării cu expresii regulate poate fi rezolvată în timpul $O(n + 2^{m_\Sigma})$ cu automate deterministe și spațiu $O(2^{m_\Sigma})$.

Theorem (Myers, 1992)

Problema căutării cu expresii regulate poate fi rezolvată în timpul $O(mn/\log n)$ cu automate deterministe și spațiu $O(mn/\log n)$.