Patterns, Regular Expressions and Finite Automata

(LECTURE 5)

Patterns and their defined languages

- S: a finite alphabet
- A pattern is a string of symbols representing a set of strings in S*.
- The set of all patterns is defined inductively as follows:

  1. atomic patterns:

     $a \in S$, e, ∅, #, @.

  2. compound patterns: if a and b are patterns, then so are: $a + b$, $a \cap b$, a^*, a^+, ~ a and $a \cdot b$.

- For each pattern a, L(a) is the language represented by a and is defined inductively as follows:

  1. L(a) = {a}, L(e) = {e}, L(∅)= {}, L(#) = S, L(@) = S *.
  2. If L(a) and L(b) have been defined, then

     $L(a + b) = L(a) \cup L(b)$, $L(a \cap b) = L(a) \cap L(b)$.

     $L(a^+) = L(a)^+$, $L(a^*) = L(a)^*$,

     $L(\sim a) = S^* - L(a)$, $L(a \cdot b) = L(a) \cdot L(b)$.

More on patterns

- We say that a string x matches a pattern a iff x ∈ L(a).
- Some examples:

1. *S= L(@) = L(#*)
2. L(x) = {x} for any x ∈ S*
3. for any $x_1,\ldots,x_n$ in S*, $L(x_1+x_2+\ldots+x_n) = \{x_1,x_2,\ldots,x_n\}$.
4. {x | x contains at least 3 a's} = L(@a@a@a@}
5. S{a} = # ∩ ~a
6. {x | x does not contain a} = (# ∩ ~a)*
7. {x | every 'a' in x is followed sometime later by a 'b' } =

   = {x | either no 'a' in x or $ 'b' in x followed no 'a' }

   = (# ∩ ~a)* + @b(# ∩ ~a)*

More on pattern matching

- Some interesting and important questions:

1. How hard is it to determine if a given input string x matches a given pattern a ?

   ==> efficient algorithm exists

2. Can every set be represented by a pattern ?

   ==> no! the set $\{a^n b^n \mid n > 0\}$ cannot be represented by any pattern.

3. How to determine if two given patterns a and b are equivalent ? (I.e., L(a) = L(b)) --- an exercise !

4. Which operations are redundant ?
   - $e = \sim(\#^+ \cap @) = \emptyset^*$; $a^+ = a \cdot a^*$
   - $\# = a_1 + a_2 + ... + a_n$ if $S = \{a_1, .., a_n\}$
   - $a + b = \sim(\sim a \cap \sim b)$; $a \cap b = \sim(\sim a + \sim b)$
   - It can be shown that ~ is redundant.

Equivalence of patterns, regular expr. & FAs

- Recall that regular expressions are those patterns that can be built from: a $\in$S, e, $\emptyset$, +, $\cdot$ and *.
- Notational conventions:
  - a + br means a + (br)
  - a + b* means a + (b*)
  - a b* means a (b*)

Theorem 8: Let A $\subseteq$ S*. Then the followings are equivalent:

1. A is regular (I.e., A = L(M) for some FA M),
2. A = L(a) for some pattern a,
3. A = L(b) for some regular expression b.

pf: Trivial part: (3) => (2).

(2) => (1) to be proved now!

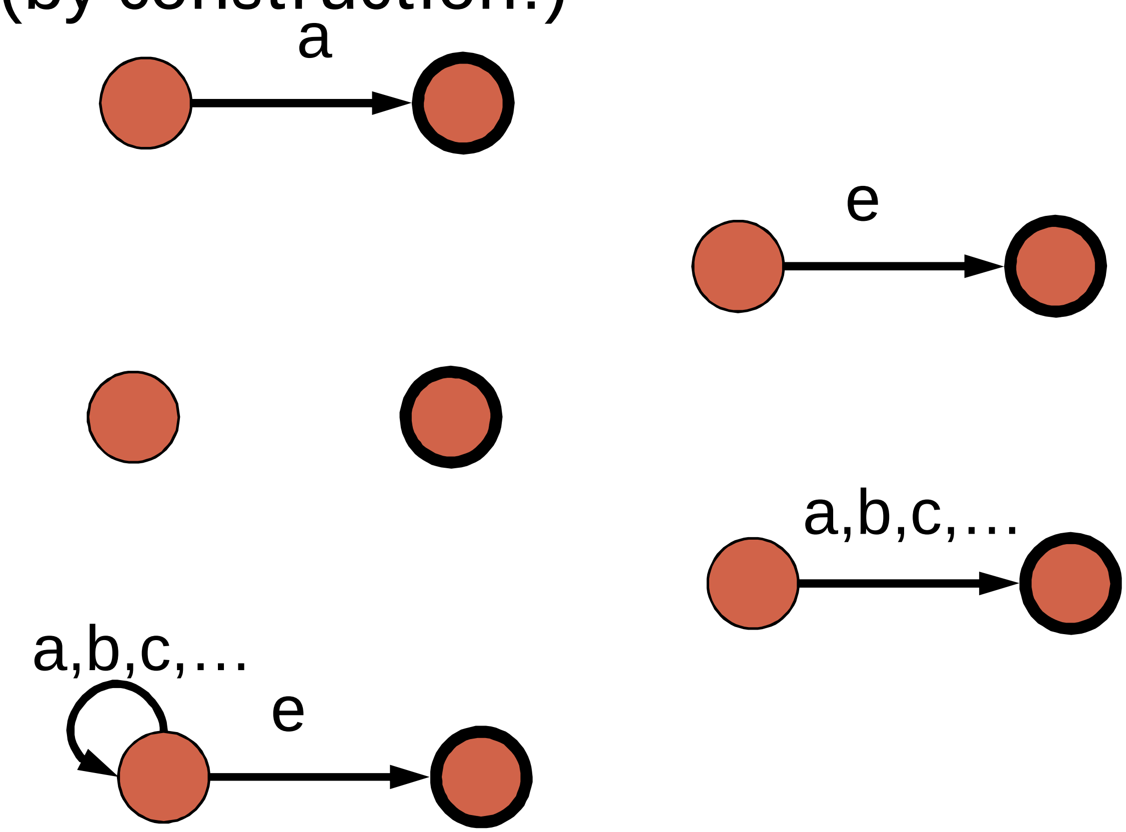
(1)=> (3) later.

(2) => (1) : Every set represented by a pattern is regular

Pf: By induction on the structure of pattern a.

Basis: a is atomic: (by construction!)

1. a = a :
2. a = e:
3. a = ∅ :
4. a = #:
5. a@ = #* :



Inductive cases: Let M_1 and M_2 be any FAs accepting L(b) and L(g), respectively.

6. æ= b g: => L(a) = L($M_1 \cdot M_2$)

7. æ= b * : => L(a) = L(M_1*)

8. æ= b + g, æ= ~b or a = b ∩ g: By ind. hyp. b and gare regular. Hence by closure properties of regular languages, a is regular, too.

9. æ= b^+ = b b* : Similar to case 8.

Some examples patterns & their equivalent FAs

1. (aaa)* + (aaaaa)*

(1)=>(3): Regular languages can be represented by reg. expr.

M = (Q, S, d, S, F) : a NFA; X⊆ Q: a set of states; m, n∈Q : two states

- $p^X(m, n)$ =$_{def}$ {y ∈ S* | $ a path from m to n labeled y and all intermediate states ∈ X }.
  - Note: L(M) = ?
- $p^X(m, n)$ can be shown to be representable by a regular expr, by induction as follows:

Let D(m,n) = { a | (m –a→n) ∈ d } = {a_1,...,a_k} (k≥ 0)

= the set of symbols by which we can reach from m to n, then

Basic case: X = ∅ :

1.1 if m ≠ n: $p^{\emptyset}(m, n)$ = {a_1, a_2,...,a_k } = L(a_1 + a_2+...+ a_k) if k > 0,

= {} = L(∅) if k = 0.

1.2 if m =n: $p^{\emptyset}(m, n)$ = {a_1, a_2,... a_k, e}=L(a_1 + a_2+...+ a_k +e) if k > 0,

= {e} = L(e) if k = 0.

3. For nonempty X, let q be any state in X, then :

$p^X(m,n) = p^{X-\{q\}}(m,n)\ U\ p^{X-\{q\}}(m,q)\ (p^{X-\{q\}}(q,q))^*\ p^{X-\{q\}}(q,n)$.

By Ind.hyp.(why?), there are regular expressions a, b, g, r with

$L([a, b, g, r]) = [p^{X-\{q\}}(m,n), p^{X-\{q\}}(m,q), (p^{X-\{q\}}(q,q)), p^{X-\{q\}}(q,n)]$

Hence $p^X(m,n) = L(a)\ U\ L(b)\ L(g)^*\ L(r)$,

$= L(a + bg^*r)$

and can be represented as a reg. expr.

- Finally, L(M) = {x | s --x--> f, s ∈ S, f ∈ F }

  $= S_{s \in S, f \in F}\ p^Q(s,f)$, is representable by a regular expression.

Some examples

Example (9.3): M :

- $L(M) = p^{\{p,q,r\}}(p,p) = p^{\{p,r\}}(p,p) + p^{\{p,r\}}(p,q)\,(p^{\{p,r\}}(q,q))^*\,p^{\{p,r\}}(q,p)$
- $p^{\{p,r\}}(p,p) = ?$
- $p^{\{p,r\}}(p,q) = ?$
- $p^{\{p,r\}}(q,q) = ?$
- $p^{\{p,r\}}(q,p) = ?$

Hence L(M) = ?

	0	1
>pF	{p}	{q}
q	{r}	{}
r	{p}	{q}

Another approach

- The previous method
  - easy to prove,
  - easy for computer implementation, but
  - hard for human computation.
- The strategy of the new method:
  - reduce the number of states in the target FA and
  - encodes path information by regular expressions on the edges.
  - until there is one or two states : one is the start state and one is the final state.

Steps

0. Assume the machine M has only one start state and one final state. Both may probably be identical.
1. While the exists a third state p that is neither start nor final:

   1.1 (Merge edges) For each pair of states (q,r) that has more than 1 edges with labels $t_1, t_2, \ldots t_n$, respectively, than merge these edges by a new one with regular expression $t = t_1 + t_2 \ldots + t_n$.

   1.2 (Replace state p by edges; remove state)

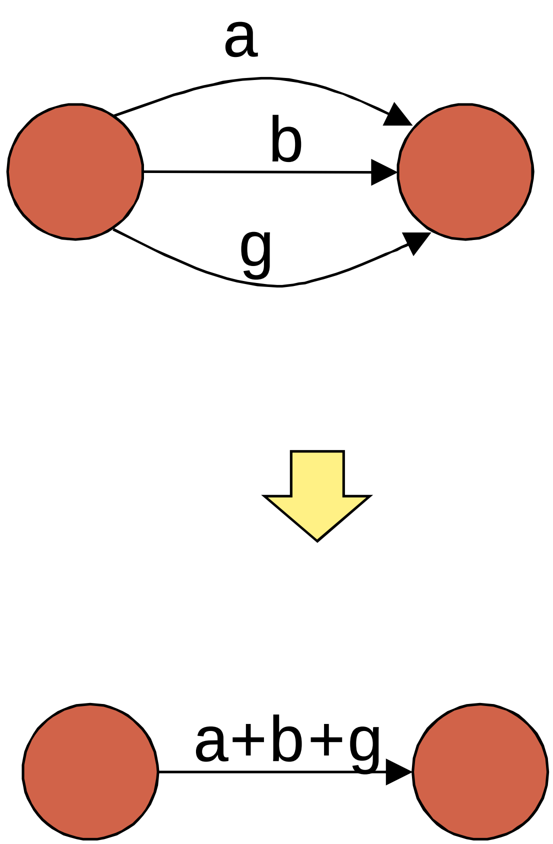
   Let $(p_1, a_1, p), \ldots (p_n, a_n, p)$ where p_j != p be the collection of all edges in M with p as the destination state, and

   $(p, b_1, q_1), \ldots, (p, b_m, q_m)$ where qj != p be the collection of all edges with p as the start state. Now the sate p together with all its connecting edges can be removed and replaced by a set of m x n new edges :

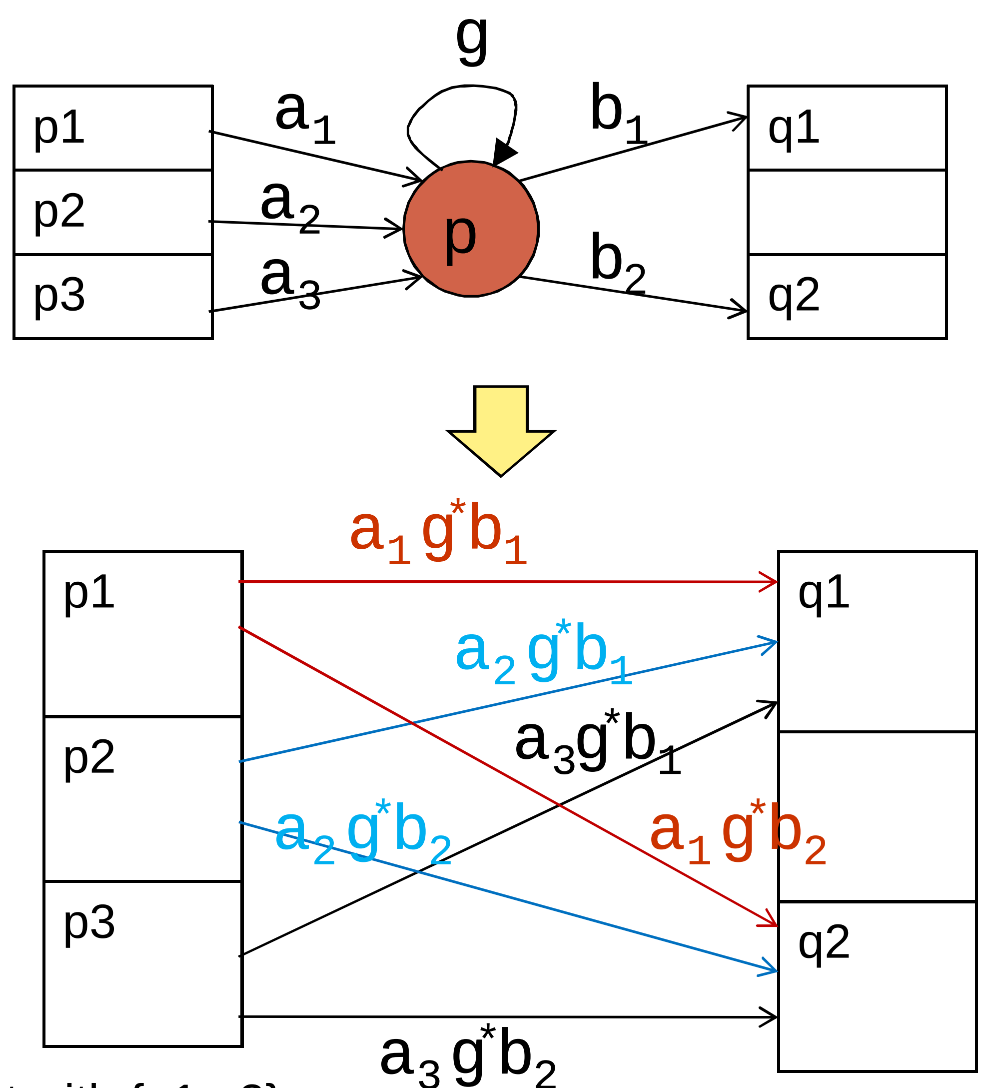
   $\{ (p_i, a_i\, t^*\, b_j, q_j) \mid i \text{ in } [1,n] \text{ and } j \text{ in } [1,m] \}$.

   The new machine is equivalent to the old one.

- Merge Edges :



•Replace state by Edges



Note: {p1,p2,p3} may intersect with {q1,q2}.

2. perform 1.1 once again (merge edges)

// There are one or two states now

3 Two cases to consider:

3.1 The final machine has only one state, that is both start and final. Then if there is an edge labeled t on the sate,

then t* is the result, other the result is e.

3.2 The machine has one start state s and one final state f.

Let (s, s→s, s), (f, f→f, f), (s,s→f, f) and (f, f→f, f) be the collection of all edges in the machine, where (s→f) means the regular expression or label on the edge from s to f.

The result then is

Example

	0	1
>p	{p,r}	{q,r}
q	{r}	{p,q,r}
rF	{p,q}	{q,r}

1. another representation

	p	q	r
p	0	1	0,1
q	1	1	0,1
r	0	0,1	1

Merge edges

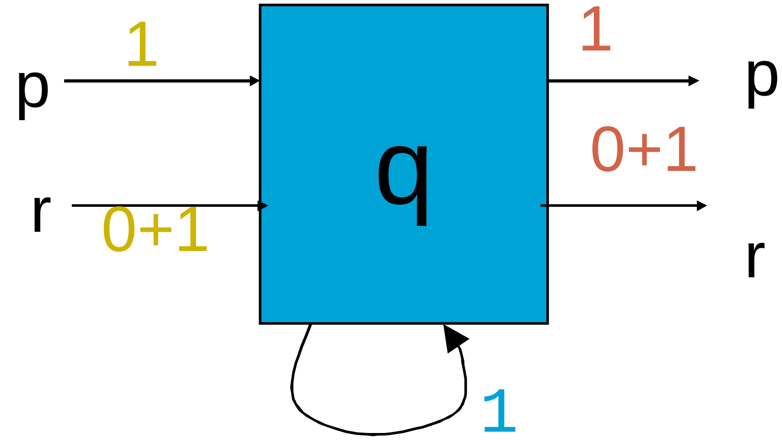
	p	q	r
p	0	1	0,1
q	1	1	0,1
r	0	0,1	1

	p	q	r
p	0	1	0+1
q	1	1	0+1
r	0	0+1	1

remove q

	p	q	r
p	0	1	0+1
q	1	1	0+1
r	0	0+1	1

	p	q	r
p	0, 11*1	1	0+1, 11* (0+1)
q	1	1,	0+1
r	0, (0+1) 1*1	0+1	1, (0+1)1*(0+1)



Form the final result

	p	r
>p	0+11*1	0+1+11* (0+1)
rF	0+ (0+1) 1*1	1+ (0+1)1*(0+1)

Final result : = [p→p + (p→r) (r→r)* (r→p)]* (p→r) (r→r) *

[(0+11*1) +(0+1+11*(0+1)) (1+(0+1)1*(0+1))* (0+(0+1)1*1)]*

(0+1+11*(0+1))(1+(0+1)1*(0+1))*