COURSE: THEORY OF AUTOMATA COMPUTATION

TOPICS TO BE COVERED

- Patterns and their defined languages
- Regular Expressions
- Finite Automata

PATTERNS AND THEIR DEFINED LANGUAGES

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

     $a \in \Sigma$, ε, $\emptyset$, #, @.
  2. compound patterns: if α and β are patterns, then so are: $\alpha + \beta$, $\alpha \cap \beta$, α^*, α^+, $\sim \alpha$ and $\alpha \cdot \beta$.
- For each pattern α, $L(\alpha)$ is the language represented by α and is defined inductively as follows:
  1. $L(a) = \{a\}$, $L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \{\}$, $L(\#) = \Sigma$, $L(@) = \Sigma^*$.
  2. If $L(\alpha)$ and $L(\beta)$ have been defined, then

     $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$, $L(\alpha \cap \beta) = L(\alpha) \cap L(\beta)$.

     $L(\alpha^+) = L(\alpha)^+$, $L(\alpha^*) = L(\alpha)^*$,

     $L(\sim \alpha) = \Sigma^* - L(\alpha)$, $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$.

MORE ON PATTERNS

- We say that a string x matches a pattern α iff $x \in L(\alpha)$.
- Some examples:

1. $\Sigma^* = L(@) = L(\#^*)$
2. $L(x) = \{x\}$ for any $x \in \Sigma^*$
3. for any $x_1,\ldots,x_n$ in Σ^*, $L(x_1+x_2+\ldots+x_n) = \{x_1,x_2,\ldots,x_n\}$.
4. {x | x contains at least 3 a's} = L(@a@a@a@}
5. $\Sigma - \{a\} = \# \cap \sim a$
6. $\{x \mid x \text{ does not contain a}\} = (\# \cap \sim a)^*$
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' }

   $= (\# \cap \sim a)^* + @b(\# \cap \sim a)^*$

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 α and β are equivalent ? (I.e., $L(\alpha) = L(\beta)$) --- an exercise !

4. Which operations are redundant ?

   - $\varepsilon = \sim(\#^+ \cap @) = \varnothing^*$; $\alpha^+ = \alpha \cdot \alpha^*$
   - $\# = a_1 + a_2 + \ldots + a_n$ if $\Sigma = \{a_1, .., a_n\}$
   - $\alpha + \beta = \sim(\sim\alpha \cap \sim\beta)$; $\alpha \cap \beta = \sim(\sim\alpha + \sim\beta)$
   - 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: $a \in \Sigma$, ε, $\varnothing$, $+$, $\cdot$ and $*$.
- Notational conventions:
  - $\alpha + \beta\rho$ means $\alpha + (\beta\rho)$
  - $\alpha + \beta^*$ means $\alpha + (\beta^*)$
  - $\alpha \beta^*$ means $\alpha (\beta^*)$

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

1. A is regular (I.e., $A = L(M)$ for some FA M),
2. $A = L(\alpha)$ for some pattern α,
3. $A = L(\beta)$ for some regular expression β.

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

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

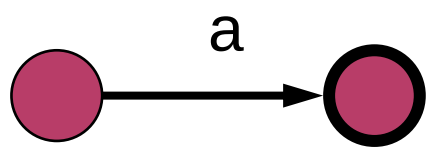
(1)=> (3) later.

(2) => (1) : EVERY SET REPRESENTED BY A PATTERN IS REGULAR

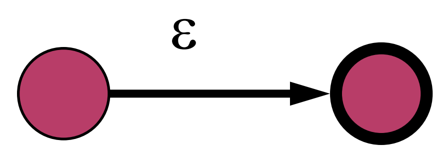
Pf: By induction on the structure of pattern α.

Basis: α is atomic: (by construction!)

1. α = a :

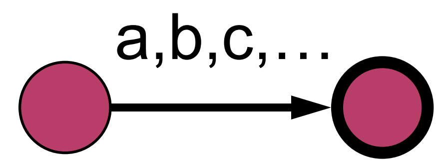


2. α = ε:

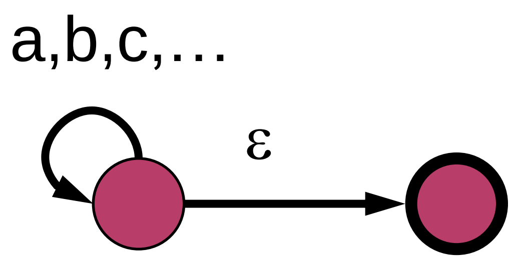


3. α = $\varnothing$:

4. α = #:



5. α = @ = #* :



Inductive cases: Let M_1 and M_2 be any FAs accepting $L(\beta)$ and $L(\gamma)$, respectively.

6. $\alpha = \beta\gamma$: => $L(\alpha) = L(M_1 \cdot M_2)$

7. $\alpha = \beta^*$: => $L(\alpha) = L(M_1{}^*)$

8. $\alpha = \beta + \gamma$, $\alpha = \sim\beta$ or $\alpha = \beta \cap \gamma$: By ind. hyp. β and γ are regular. Hence by closure properties of regular languages, α is regular, too.

9. $\alpha = \beta^+ = \beta\beta^*$: Similar to case 8.

SOME EXAMPLES PATTERNS & THEIR EQUIVALENT FAS

1. (aaa)* + (aaaaa)*

(1)=>(3): REGULAR LANGUAGES CAN BE REPRESENTED BY REG. EXPR.

$M = (Q, \Sigma, \delta, S, F)$: a NFA; $X \subseteq Q$: a set of states; $\mu, \nu \in Q$: two states

- $\pi^X(\mu,\nu) =_{def} \{y \in \Sigma^* \mid \exists$ a path from μ to ν labeled y and all intermediate states $\in X \}$.
  - Note: $L(M) = ?$
- $\pi^X(\mu,\nu)$ can be shown to be representable by a regular expr, by induction as follows:

Let $D(\mu,\nu) = \{ a \mid (\mu \text{ -}a\rightarrow \nu) \in \delta \} = \{a_1,\ldots,a_k\}$ ($k \geq 0$)

= the set of symbols by which we can reach from μ to ν, then

Basic case: $X = \varnothing$:

1.1 if $\mu \neq \nu$: $\pi^{\varnothing}(\mu,\nu) = \{a_1, a_2,\ldots,a_k \} = L(a_1 + a_2+\ldots+ a_k)$ if $k > 0$,

$= \{\} = L(\varnothing)$ if $k = 0$.

1.2 if $\mu = \nu$: $\pi^{\varnothing}(\mu,\nu) = \{a_1, a_2,\ldots a_k, \varepsilon\}=L(a_1 + a_2+\ldots+ a_k +\varepsilon)$ if $k > 0$,

$= \{\varepsilon\} = L(\varepsilon)$ if $k = 0$.

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

$\pi^X(\mu,\nu) = \pi^{X-\{q\}}(\mu,\nu) \cup \pi^{X-\{q\}}(\mu,q) \; (\pi^{X-\{q\}}(q,q))^* \; \pi^{X-\{q\}}(q,\nu)$.

By Ind.hyp.(why?), there are regular expressions $\alpha, \beta, \gamma, \rho$ with

$L([\alpha, \beta, \gamma, \rho]) = [\pi^{X-\{q\}}(\mu,\nu), \pi^{X-\{q\}}(\mu,q), (\pi^{X-\{q\}}(q,q)), \pi^{X-\{q\}}(q,\nu)]$

Hence $\pi^X(\mu,\nu) = L(\alpha) \cup L(\beta) \; L(\gamma)^* \; L(\rho)$,

$= L(\alpha + \beta\gamma^*\rho)$

and can be represented as a reg. expr.

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

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

SOME EXAMPLES

Example : 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)

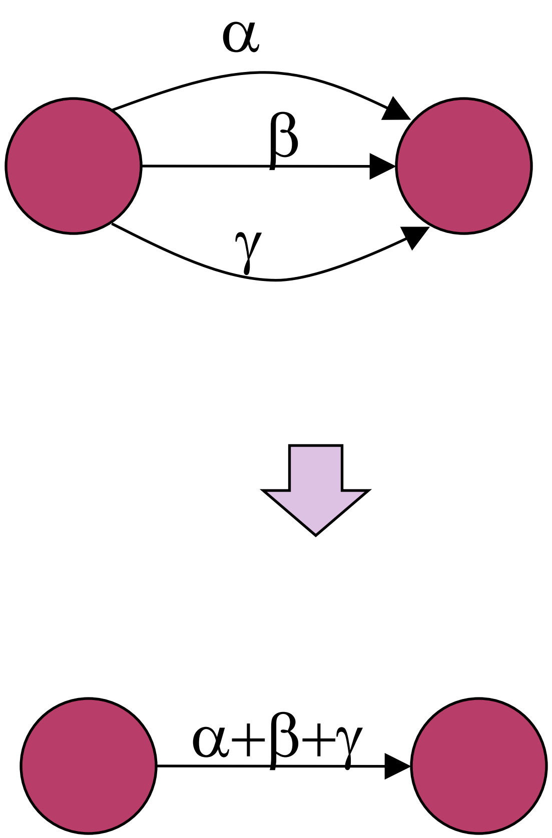
   Let $(p_1, \alpha_1, p), \ldots (p_n, \alpha_n, p)$ where p_j != p be the collection of all edges in M with p as the destination state, and

   $(p, \beta_1, q_1), \ldots, (p, \beta_m, q_m)$ 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 :

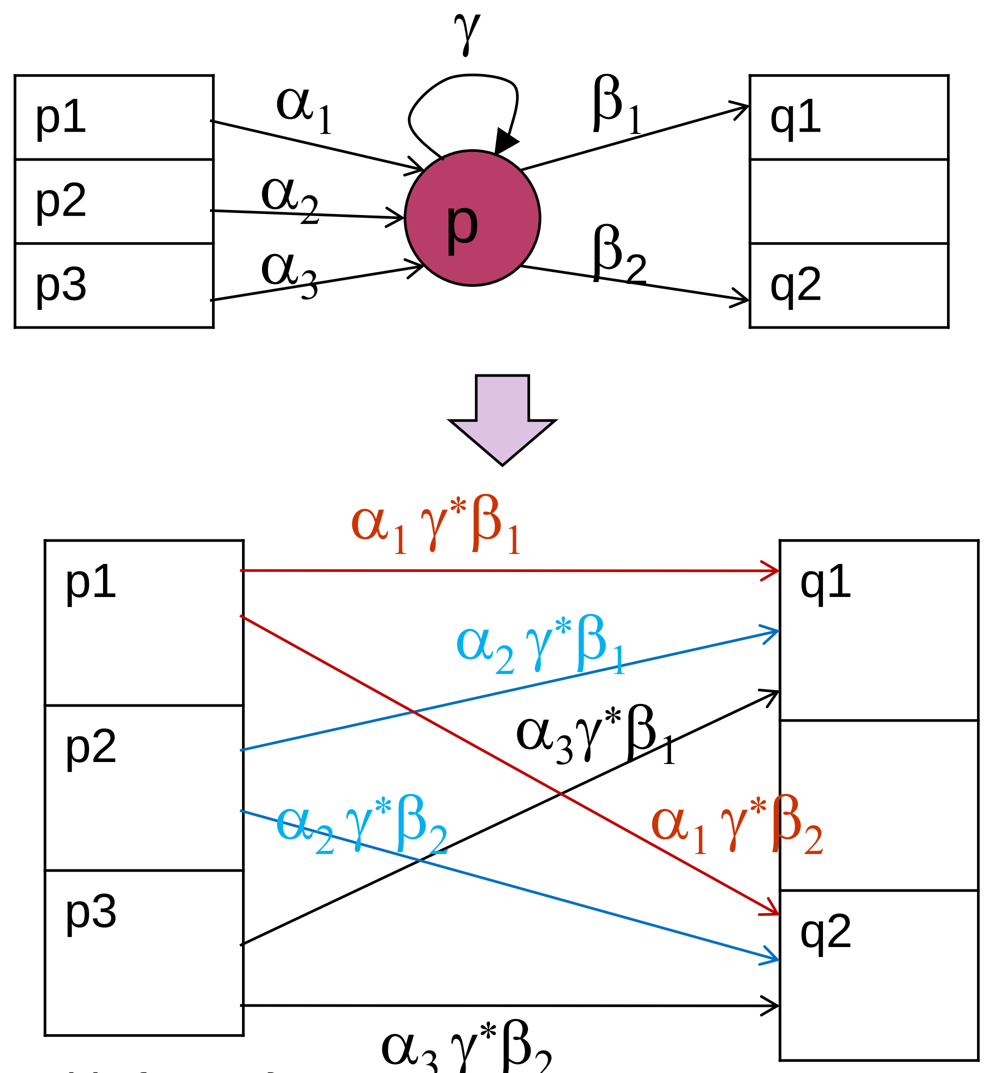
   $\{ (p_i, \alpha_i t^* \beta_j, q_j) \mid i \text{ in } [1,n] \text{ and } j \text{ in } [1,m] \}$.

   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 ε.

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

$$[\ (s\to s) + (s\to f\ )\ (f\to f)^*\ (f\to s)\ ]^*\ (s\to f)\ (f\to f)^*$$

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

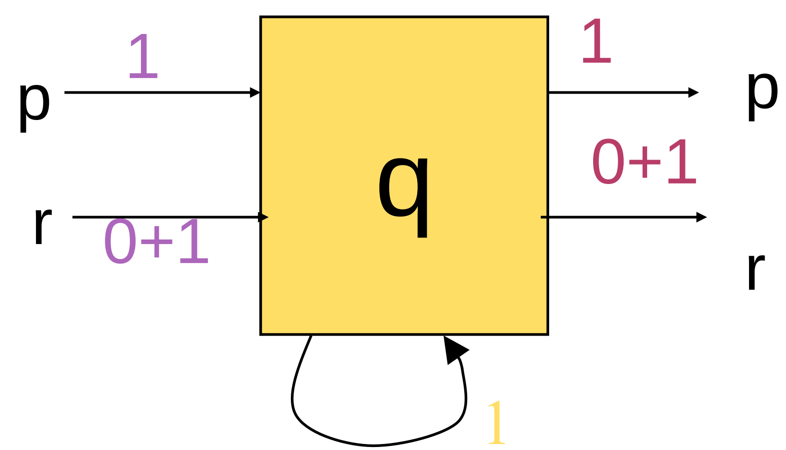
	p	q	r
p	0	1	0,1
q	1	1	0,1
r	0	0,1	1

	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, 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)

	p	q	r
p	0	1	0+1
q	1	1	0+1
r	0	0+1	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))*