CS310M: Automata Theory (Minor)

Topic 4: Regular Expressions

Paritosh Pandya

Indian Institute of Technology, Bombay
Course URL: https://cse.iitb.ac.in/∼pandya58/CS310M/automata.html

Autumn, 2021

Specifying Patterns of Occurrence of Letters

- We define patterns of occurrence of symbols from Σ using pattern expressions.
  E.g a^*b denotes zero or more 'a' followed by one 'b'.

Next we define the pattern expressions and their language.

Specifying Patterns of Occurrence of Letters

- We define patterns of occurrence of symbols from Σ using pattern expressions.
  E.g a^*b denotes zero or more 'a' followed by one 'b'.
- We will use α, β, γ to denote pattern expressions.

Next we define the pattern expressions and their language.

Specifying Patterns of Occurrence of Letters

- We define patterns of occurrence of symbols from Σ using pattern expressions.
  E.g a^*b denotes zero or more 'a' followed by one 'b'.
- We will use α, β, γ to denote pattern expressions.
- A word x in Σ^* may match a pattern α. This is denoted by $x \in L(\alpha)$. Thus $L(\alpha)$ is collection of words matching α.
  E.g. $L(a^*b) = \{a^n \cdot b \mid n \geq 0\}$.

Next we define the pattern expressions and their language.

Specifying Patterns of Occurrence of Letters

- We define patterns of occurrence of symbols from Σ using pattern expressions.
  E.g a^*b denotes zero or more 'a' followed by one 'b'.
- We will use α, β, γ to denote pattern expressions.
- A word x in Σ^* may match a pattern α. This is denoted by $x \in L(\alpha)$. Thus $L(\alpha)$ is collection of words matching α.
  E.g. $L(a^*b) = \{a^n \cdot b \mid n \geq 0\}$.
- E.g. $L(a^* \cdot b + b \cdot a^*) = \{a^n \cdot b,\ b \cdot a^n \mid n \geq 0\}$

Next we define the pattern expressions and their language.

Atomic Patterns over alphabet Σ

- ϕ, No word matches. $L(\phi) = \emptyset$.

Atomic Patterns over alphabet Σ

- ϕ, No word matches. $L(\phi) = \emptyset$.
- a matches the single letter $a \in \Sigma$. $L(a) = \{a\}$

Atomic Patterns over alphabet Σ

- ϕ, No word matches. $L(\phi) = \emptyset$.
- a matches the single letter $a \in \Sigma$. $L(a) = \{a\}$
- ϵ Only empty word matches. $L(\epsilon) = \{\epsilon\}$.

Atomic Patterns over alphabet Σ

- ϕ, No word matches. $L(\phi) = \emptyset$.
- a matches the single letter $a \in \Sigma$. $L(a) = \{a\}$
- ϵ Only empty word matches. $L(\epsilon) = \{\epsilon\}$.
- $\#$. Matches any letter from Σ. $L(\#) = \Sigma$

Atomic Patterns over alphabet Σ

- ϕ, No word matches. $L(\phi) = \emptyset$.
- a matches the single letter $a \in \Sigma$. $L(a) = \{a\}$
- ϵ Only empty word matches. $L(\epsilon) = \{\epsilon\}$.
- $\#$ Matches any letter from Σ. $L(\#) = \Sigma$
- @ Matches any word from Σ. $L(@) = \Sigma^*$

Compound Patterns

We use operations like catenation, intersection on languages to get more complex patterns.

- $(\alpha + \beta)$. Union. $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$.

Compound Patterns

We use operations like catenation, intersection on languages to get more complex patterns.

- $(\alpha + \beta)$. Union. $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$.
- $(\alpha \cdot \beta)$. Catenation. $L((\alpha \cdot \beta)) = L(\alpha) \cdot L(\beta)$.
  We will typically write $\alpha \cdot \beta$ as $\alpha\beta$.

Compound Patterns

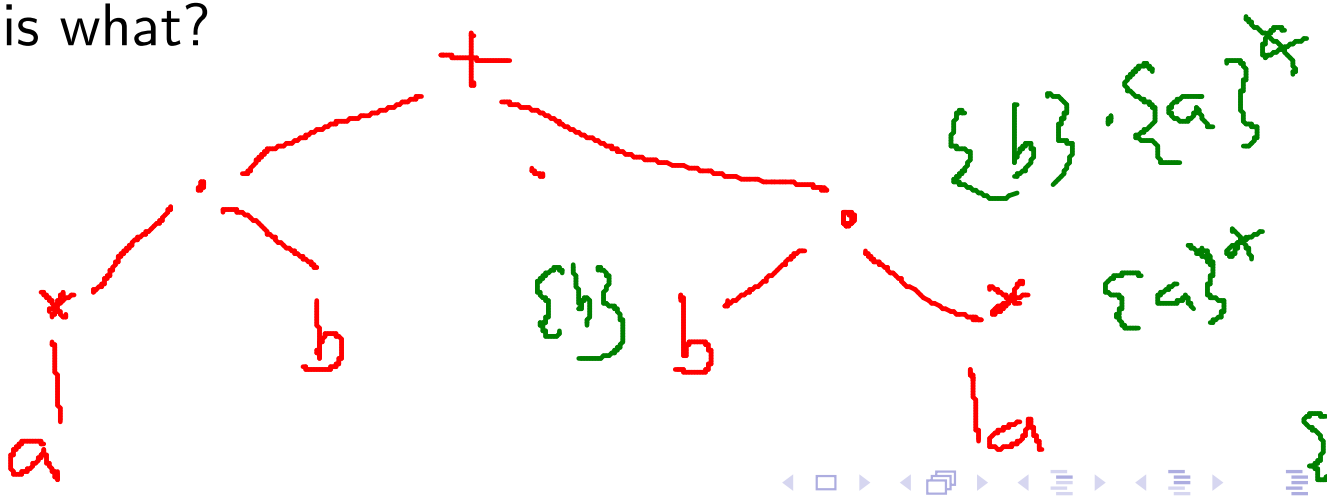
We use operations like catenation, intersection on languages to get more complex patterns.

- $(\alpha + \beta)$. Union. $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$.
- $(\alpha \cdot \beta)$. Catenation. $L((\alpha \cdot \beta)) = L(\alpha) \cdot L(\beta)$.
  We will typically write $\alpha \cdot \beta$ as $\alpha\beta$.
- (α^*). Kleene Closure.
  $L((\alpha^*)) = L(\alpha)^* = \bigcup_{n \geq 0} L(\alpha)^n$.

Compound Patterns

We use operations like catenation, intersection on languages to get more complex patterns.

- $(\alpha + \beta)$. Union. $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$.
- $(\alpha \cdot \beta)$. Catenation. $L((\alpha \cdot \beta)) = L(\alpha) \cdot L(\beta)$.
  We will typically write $\alpha \cdot \beta$ as $\alpha\beta$.
- (α^*). Kleene Closure.
  $L((\alpha^*)) = L(\alpha)^* = \bigcup_{n \geq 0} L(\alpha)^n$.
- Example: Compound Pattern $(((a^*) \cdot b) + (b \cdot (a^*)))$. Its language is what?



Compound Patterns

We use operations like catenation, intersection on languages to get more complex patterns.

- $(\alpha + \beta)$. Union. $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$.
- $(\alpha \cdot \beta)$. Catenation. $L((\alpha \cdot \beta)) = L(\alpha) \cdot L(\beta)$.
  We will typically write $\alpha \cdot \beta$ as $\alpha\beta$.
- (α^*). Kleene Closure.
  $L((\alpha^*)) = L(\alpha)^* = \bigcup_{n \geq 0} L(\alpha)^n$.
- Example: Compound Pattern $(((a^*) \cdot b) + (b \cdot (a^*)))$. Its language is what?
- $(\alpha \cap \beta)$. Intersection. $L((\alpha \cap \beta)) = L(\alpha) \cap L(\beta)$.

Compound Patterns

We use operations like catenation, intersection on languages to get more complex patterns.

- $(\alpha + \beta)$. Union. $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$.
- $(\alpha \cdot \beta)$. Catenation. $L((\alpha \cdot \beta)) = L(\alpha) \cdot L(\beta)$.
  We will typically write $\alpha \cdot \beta$ as $\alpha\beta$.
- (α^*). Kleene Closure.
  $L((\alpha^*)) = L(\alpha)^* = \bigcup_{n \geq 0} L(\alpha)^n$.
- Example: Compound Pattern $(((a^*) \cdot b) + (b \cdot (a^*)))$. Its language is what?
- $(\alpha \cap \beta)$. Intersection. $L((\alpha \cap \beta)) = L(\alpha) \cap L(\beta)$.
- $(\sim \alpha)$. Complement.
  $L((\sim \alpha)) = \sim L(\alpha) = \Sigma^* - L(\alpha)$.

Example of Pattern expression and its langauge

Let α be the pattern $(((a^*) \cdot b) + (b \cdot (a^*)))$. Then $L(\alpha)$ is what?

Inductive Definition

- Every pattern expression is uniquely constructed from atomic patterns by successive application of compound operators.
- The parse tree makes clear the structure of the expression.
- THe language of pattern expression can be computed in bottom up fashion by applying rules for each node.

This method is quite general. Also used to define programming languages.

Reducing Brackets: Operator Precedence

- Similar to operator precedence rules in arithmetic expressions: $a + b * c$ is $(a + (b * c))$

Reducing Brackets: Operator Precedence

- Similar to operator precedence rules in arithmetic expressions: $a + b * c$ is $(a + (b * c))$
- Precedence of operators: $* > \cdot > \sim > \cap > +$.

Reducing Brackets: Operator Precedence

- Similar to operator precedence rules in arithmetic expressions: $a + b * c$ is $(a + (b * c))$
- Precedence of operators: $* > \cdot > \sim > \cap > +$.
- Associativity: $((\alpha + \beta) + \gamma) \equiv (\alpha + (\beta + \gamma))$. Hence we can write $\alpha + \beta + \gamma$.
  Similarly, $((\alpha\beta)\gamma) \equiv (\alpha(\beta\gamma))$. Hence $\alpha\beta\gamma$.

Reducing Brackets: Operator Precedence

- Similar to operator precedence rules in arithmetic expressions: $a + b * c$ is $(a + (b * c))$
- Precedence of operators: $* > \cdot > \sim > \cap > +$.
- Associativity: $((\alpha + \beta) + \gamma) \equiv (\alpha + (\beta + \gamma))$. Hence we can write $\alpha + \beta + \gamma$.
  Similarly, $((\alpha\beta)\gamma) \equiv (\alpha(\beta\gamma))$. Hence $\alpha\beta\gamma$.
- Example $(((a^*) \cdot b) + (b \cdot (a^*)))$ can be written as $a^*b + ba^*$.

Examples

- $L(@) = \Sigma^* = L(\#^*)$
- a^*b^*
- $@a@a@a@$ gives words with three occurrences of a
- $(\#\cap a)^*$ strings without letter a.
- Sentences having "cat" and "nation" in it.
  $(@cat@) \cap (@nation@)$.
  E.g. The nation of cats. and The catenation of two words is a word.

Examples

- $L(@) = \Sigma^* = L(\#^*)$
- a^*b^*
- $@a@a@a@$ gives words with three occurrences of a
- $(\#a \cap a)^*$ strings without letter a.
- Sentences having "cat" and "nation" in it.
  $(@cat@) \cap (@nation@)$.
  E.g. The nation of cats. and The catenation of two words is a word.

Equivalence of Patterns

Define $\alpha \equiv \beta$ iff $L(\alpha) = L(\beta)$.

Examples

- $L(@) = \Sigma^* = L(\#^*)$
- a^*b^*
- $@a@a@a@$ gives words with three occurrences of a
- $(\#a \cap a)^*$ strings without letter a.
- Sentences having "cat" and "nation" in it.
  $(@cat@) \cap (@nation@)$.
  E.g. The nation of cats. and The catenation of two words is a word.

Equivalence of Patterns

Define $\alpha \equiv \beta$ iff $L(\alpha) = L(\beta)$.

Example $(a+b)c \equiv ac + bc$.

Redundant Operators

- Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$. Then $\# \equiv (a_1 + a_2 + \ldots + a_n)$.
- $@ \equiv \#^*$.
- $\alpha \cap \beta \equiv \sim(\sim \alpha \cup \sim \beta)$.
- $\sim$ is also redundant (difficult proof).

Regular Expressions

Redundant Operators

- Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$. Then $\# \equiv (a_1 + a_2 + \ldots + a_n)$.
- $@ \equiv \#^*$.
- $\alpha \cap \beta \equiv \sim(\sim \alpha \cup \sim \beta)$.
- $\sim$ is also redundant (difficult proof).

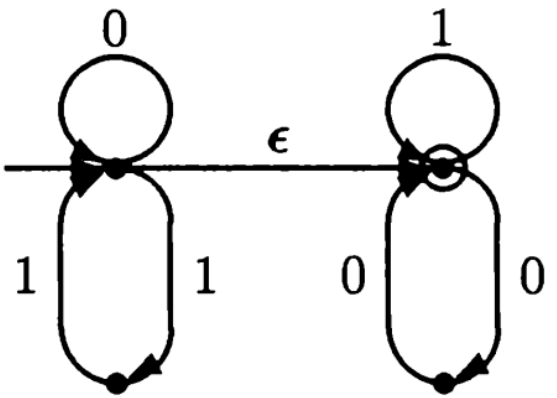
Regular Expressions

Pattern expressins without $@, \#, \cap, \sim$. Abstract syntax: let α, β range over *RegExp*.

$$\alpha ::= a \mid \phi \mid \epsilon \mid \alpha \cdot \beta \mid \alpha + \beta \mid \alpha^*$$

Question What kind of languages are definable using regular expressions? Can they be accepted by a DFA?

$$(11+0)^*(00+1)^*$$



In general?

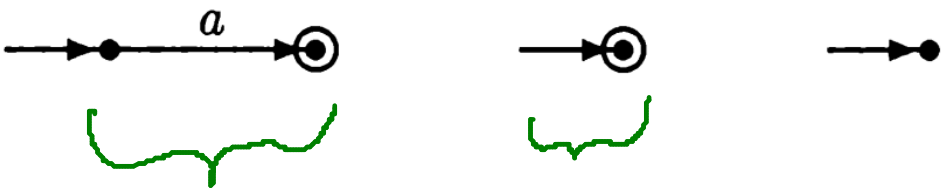
From RegExp to ϵ-NFA

Theorem

For every regular expression α we can construct a normalized ϵ-NFA $A(\alpha)$ such that $L(\alpha) = L(A(\alpha))$.

Proof (By stuctural induction). We construct the automaton bottom up.

- Normalized ϵ-NFA for a, ϵ, ϕ respectively.



From RegExp to ϵ-NFA

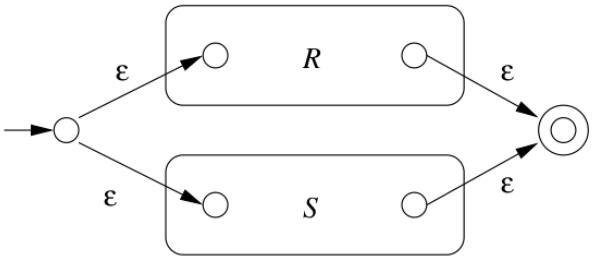
Theorem

For every regular expression α we can construct a normalized ϵ-NFA $A(\alpha)$ such that $L(\alpha) = L(A(\alpha))$.

Proof (By stuctural induction). We construct the automaton bottom up.

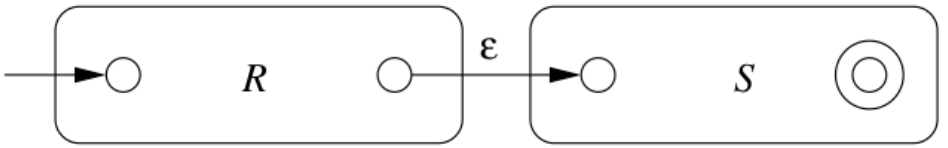
- Normalized ϵ-NFA for a, ϵ, ϕ respectively.

- For $\alpha + \beta$, inductively assume that we have normalized ϵ-NFA $A(\alpha)$ and $A(\beta)$. Construct union automaton recognizing $L(A(\alpha)) \cup L(A(\beta))$ as before.

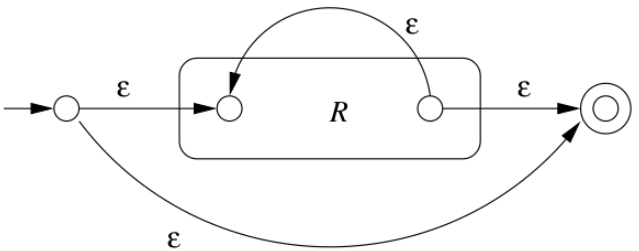


Proof (cont)

- For $\alpha \cdot \beta$, inductively assume that we have normalized ϵ-NFA $A(\alpha)$ and $A(\beta)$. Construct catenation automaton recognizing $L(A(\alpha)) \cdot L(A(\beta))$ as before.



- For α^*, inductively assume that we have normalized ϵ-NFA $A(\alpha)$. Construct Kleene closure automaton recognizing $L(A(\alpha))^*$ as before.



Size of constructed automaton

- size of regular expression α counts number of atomic expressions and number of composition operators in α. This is denoted $|\alpha|$.
  E.g. $|(a+b)^*aaa| = 11$.
- Claim The size $|A(\alpha)| = O(|\alpha|)$. How?

Size of constructed automaton

- size of regular expression α counts number of atomic expressions and number of composition operators in α. This is denoted $|\alpha|$.
  E.g. $|(a+b)^*aaa| = 11$.
- Claim The size $|A(\alpha)| = O(|\alpha|)$. How?
  Prove by induction on the structure of regular expression α that $|A(\alpha)| \leq 2 * |\alpha|$.

Size of constructed automaton

- size of regular expression α counts number of atomic expressions and number of composition operators in α. This is denoted $|\alpha|$.
  E.g. $|(a+b)^*aaa| = 11$.
- Claim The size $|A(\alpha)| = O(|\alpha|)$. How?
  Prove by induction on the structure of regular expression α that $|A(\alpha)| \leq 2 * |\alpha|$.
  - (Base step) For α equals $a \mid \phi \mid \epsilon$ check that $|A(\alpha)| \leq 2$.

Size of constructed automaton

- size of regular expression α counts number of atomic expressions and number of composition operators in α. This is denoted $|\alpha|$.
  E.g. $|(a+b)^*aaa| = 11$.
- Claim The size $|A(\alpha)| = O(|\alpha|)$. How?
  Prove by induction on the structure of regular expression α that $|A(\alpha)| \leq 2 * |\alpha|$.
  - (Base step) For α equals $a \mid \phi \mid \epsilon$ check that $|A(\alpha)| \leq 2$.
  - (Induction step 1) Let $\alpha = (\beta + \gamma)$.

$$
\begin{aligned}
|A(\alpha)| &= 2 + |A(\beta)| + |A(\gamma)| && \text{(const. } A(\alpha)\text{)},\\
&\leq 2 + 2 * |\beta| + 2 * |\gamma|, && \text{(ind. hyp.)}\\
&= 2(|\alpha|). && \text{(defn of } |\alpha|\text{)}.
\end{aligned}
$$

Size of constructed automaton

- size of regular expression α counts number of atomic expressions and number of composition operators in α. This is denoted $|\alpha|$.
  E.g. $|(a+b)^*aaa| = 10$ (+, *, 3 ·), 5 vars
- Claim The size $|A(\alpha)| = O(|\alpha|)$. How?
  Prove by induction on the structure of regular expression α that $|A(\alpha)| \leq 2 * |\alpha|$.
  - (Base step) For α equals $a \mid \phi \mid \epsilon$ check that $|A(\alpha)| \leq 2$.
  - (Induction step 1) Let $\alpha = (\beta + \gamma)$.
    $$\begin{aligned} |A(\alpha)| &= 2 + |A(\beta)| + |A(\gamma)| && \text{(const. } A(\alpha)\text{)}, \\ &\leq 2 + 2 * |\beta| + 2 * |\gamma|, && \text{(ind. hyp.)} \\ &= 2(|\alpha|). && \text{(defn of } |\alpha|\text{)}. \end{aligned}$$
  - Similar induction steps for $\beta \cdot \gamma$ and β^*

Automata to RegExp

Theorem

For every NFA $A = (Q, \Sigma, \delta, I, F)$*, we can construct a langauge equivalent regular expression* $reg(A)$*.*

Automata to RegExp

Theorem

For every NFA $A = (Q, \Sigma, \delta, I, F)$, we can construct a langauge equivalent regular expression $reg(A)$.

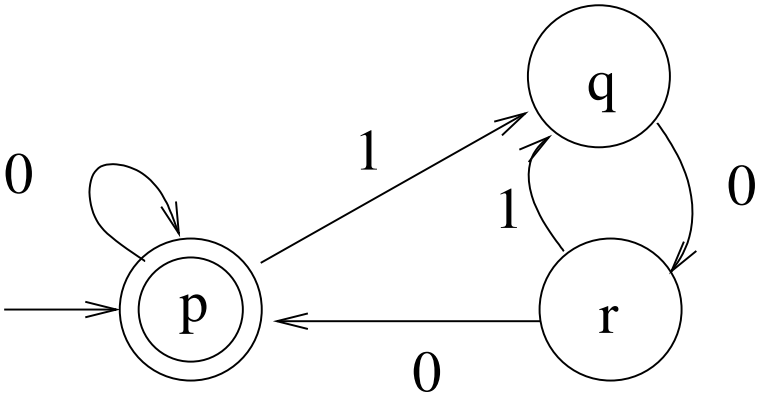
Construction Define regular expression $\alpha^X_{p,q}$ for $X \subseteq Q$ and $p, q \in Q$. This denotes set of words w such that A has a run on w from p to q where all intermediate states are from X.

Automata to RegExp

Theorem

For every NFA $A = (Q, \Sigma, \delta, I, F)$, we can construct a langauge equivalent regular expression $reg(A)$.

Construction Define regular expression $\alpha^X_{p,q}$ for $X \subseteq Q$ and $p, q \in Q$. This denotes set of words w such that A has a run on w from p to q where all intermediate states are from X.



Example:

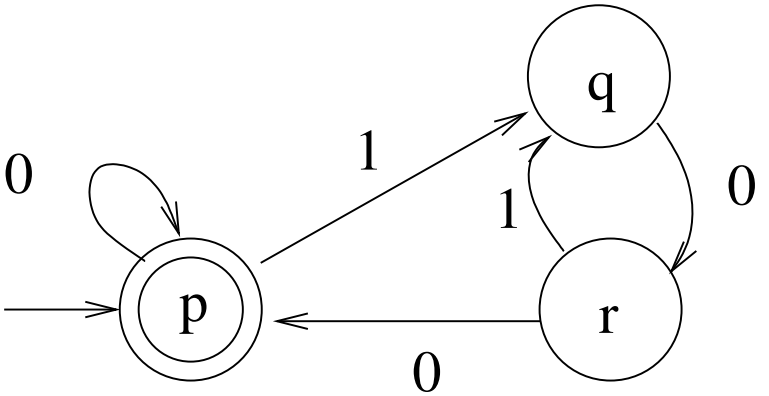
$\alpha^q_{p,r} = 1 \cdot 0$

Automata to RegExp

Theorem

For every NFA $A = (Q, \Sigma, \delta, I, F)$, we can construct a langauge equivalent regular expression $reg(A)$.

Construction Define regular expression $\alpha^X_{p,q}$ for $X \subseteq Q$ and $p, q \in Q$. This denotes set of words w such that A has a run on w from p to q where all intermediate states are from X.



Example:

$\alpha^q_{p,r} = 1 \cdot 0$ $\qquad$ $\alpha^{p,q}_{p,r} = 0^* \cdot 1 \cdot 0$

Automata to RegExp

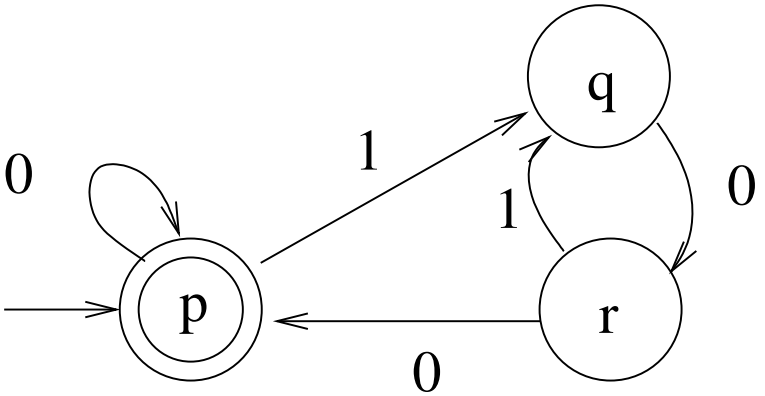
(Mcnaughton - Yamada Lemma)

Theorem

For every NFA $A = (Q, \Sigma, \delta, I, F)$, we can construct a langauge equivalent regular expression $reg(A)$.

Construction Define regular expression $\alpha^X_{p,q}$ for $X \subseteq Q$ and $p, q \in Q$. This denotes set of words w such that A has a run on w from p to q where all intermediate states are from X.

(via some sort of DP)



Example:

$\alpha^q_{p,r} = 1 \cdot 0$ $\qquad$ $\alpha^{p,q}_{p,r} = 0^* \cdot 1 \cdot 0$

Aim: To compute $\alpha^{p,q,r}_{p,p}$

Automata to RegExp (2)

$\alpha_{p,r}^{X}$ can be recursively defined using the following rules:

Automata to RegExp (2)

$\alpha^{X}_{p,r}$ can be recursively defined using the following rules:

- $\alpha^{\emptyset}_{p,r} = a_1 + \ldots + a_k$
  where $r \in \delta(p, a_i)$ provided $p \neq r$

Automata to RegExp (2)

$\alpha^X_{p,r}$ can be recursively defined using the following rules:

- $\alpha^{\emptyset}_{p,r} = a_1 + \ldots + a_k$
  where $r \in \delta(p, a_i)$ provided $p \neq r$
- $\alpha^{\emptyset}_{p,p} = a_1 + \ldots + a_k + \epsilon$
  where $p \in \delta(p, a_i)$

Automata to RegExp (2)

$\alpha^X_{p,r}$ can be recursively defined using the following rules:

- $\alpha^{\emptyset}_{p,r} = a_1 + \ldots + a_k$ where $r \in \delta(p, a_i)$ provided $p \neq r$ — direct edge
- $\alpha^{\emptyset}_{p,p} = a_1 + \ldots + a_k + \epsilon$ where $p \in \delta(p, a_i)$
- For any $q \in X$,

$$\alpha^X_{p,r} = \alpha^{(X-\{q\})}_{p,r} + \alpha^{(X-\{q\})}_{p,q} \cdot \left(\alpha^{(X-\{q\})}_{q,q}\right)^* \cdot \alpha^{(X-\{q\})}_{q,r}$$

Automata to RegExp (2)

$\alpha^X_{p,r}$ can be recursively defined using the following rules:

- $\alpha^{\emptyset}_{p,r} = a_1 + \ldots + a_k$ where $r \in \delta(p, a_i)$ provided $p \neq r$
- $\alpha^{\emptyset}_{p,p} = a_1 + \ldots + a_k + \epsilon$ where $p \in \delta(p, a_i)$
- For any $q \in X$,

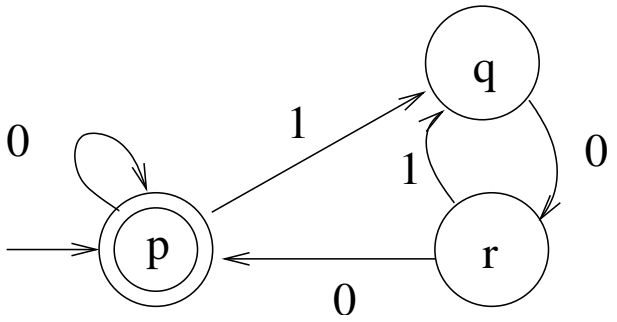
$$\alpha^X_{p,r} = \alpha^{(X-\{q\})}_{p,r} + \alpha^{(X-\{q\})}_{p,q} \cdot \left(\alpha^{(X-\{q\})}_{q,q}\right)^* \cdot \alpha^{(X-\{q\})}_{q,r}$$

The desired regular expression is given as follows:

$$\sum_{p \in I} \sum_{q \in F} \alpha^Q_{p,q}.$$

Example

automaton → regex $O(4^n)$



Compute $\alpha_{p,p}^{p,q,r}$ as

$$\alpha_{p,p}^{p,q,r} = \alpha_{p,p}^{p,r} + \alpha_{p,q}^{p,r} \cdot (\alpha_{q,q}^{p,r})^* \cdot \alpha_{q,p}^{p,r}.$$

It is easy to see that

$$\alpha_{p,p}^{p,r} = 0^*, \qquad \alpha_{p,q}^{p,r} = 0^* \cdot 1$$
$$\alpha_{q,p}^{p,r} = 00(0^*), \qquad \alpha_{q,q}^{p,r} = \epsilon + 01 + 00(0^*)1$$

Hence $\alpha_{p,p}^{p,q,r}$

$$= 0^* + 0^*1 \cdot (\epsilon + 01 + 00(0^*)1)^* \cdot 00(0^*)$$

} exponential blowup.

every exp → a tuple exp

Kleene Algebra (summary)

simplification of regular expression (language equivalent regex)

$\alpha = \beta$ iff $L(\alpha) = L(\beta)$

$\alpha \leq \beta$ iff $L(\alpha) \subseteq L(\beta)$

Algebraic Laws for equivalence of regular expressions. Useful in simplifying regexp.

- $\equiv$ is reflexive, symmetric and transitive.
- $\equiv$ is a congruence.
- $(Regexp, +, \phi)$ is a commutative monoid with $+$ idempotent.
- $(Regexp, \cdot, \epsilon)$ is a monoid.
- Distributivity: $(\alpha + \beta)\gamma \equiv (\alpha\gamma + \beta\gamma)$ and $\gamma(\alpha + \beta) \equiv (\gamma\alpha + \gamma\beta)$
- Rules for Kleene closure α^*. See Kozen.

Kleene Algebra (1)

$$\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma$$

$$\alpha + \beta \equiv \beta + \alpha$$

$$\alpha + \emptyset \equiv \alpha$$

$$\alpha + \alpha \equiv \alpha$$

$$\alpha(\beta\gamma) \equiv (\alpha\beta)\gamma$$

$$\epsilon\alpha \equiv \alpha\epsilon \equiv \alpha$$

$$\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma$$

$$(\alpha + \beta)\gamma \equiv \alpha\gamma + \beta\gamma$$

$$\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset$$

Kleene Algebra Example

$$
\begin{aligned}
(1+01+001)^*(\epsilon+0+00) &\equiv ((\epsilon+0+00)1)^*(\epsilon+0+00) \\
&\equiv ((\epsilon+0)(\epsilon+0)1)^*(\epsilon+0)(\epsilon+0).
\end{aligned}
$$

Kleene Algebra (2)

$$\epsilon + \alpha\alpha^* \equiv \alpha^*$$
$$\epsilon + \alpha^*\alpha \equiv \alpha^*$$
$$\beta + \alpha\gamma \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma$$
$$\beta + \gamma\alpha \leq \gamma \Rightarrow \beta\alpha^* \leq \gamma$$

laws are sound and complete

Soundness : $\alpha \equiv_A \beta \Rightarrow L(\alpha) = L(\beta)$

Completeness : $L(\alpha) = L(\beta) \Rightarrow \alpha \equiv_A \beta$

Useful Derived Identities

$$(\alpha\beta)^*\alpha \equiv \alpha(\beta\alpha)^*$$
$$(\alpha^*\beta)^*\alpha^* \equiv (\alpha + \beta)^*$$
$$\alpha^*(\beta\alpha^*)^* \equiv (\alpha + \beta)^*$$
$$(\epsilon + \alpha)^* \equiv \alpha^*$$
$$\alpha\alpha^* \equiv \alpha^*\alpha$$

Kleene Algebra Example (2)

$|RE| \to O(n)$

$|A(\alpha)| = ?$

$\varepsilon RE \mapsto O(n)$

$|A(\alpha)| = O(2^{2^{\cdots^{2}}} n)$ ← lower bound.

$$
\begin{aligned}
& 0^* + 0^*1(\epsilon + 00^*1)^*000^* \\
& \equiv 0^* + 0^*1(00^*1)^*000^* && \text{by (9.17)} \\
& \equiv \epsilon + 00^* + 0^*10(0^*10)^*00^* && \text{by (9.10) and (9.14)} \\
& \equiv \epsilon + (\epsilon + 0^*10(0^*10)^*)00^* && \text{by (9.8)} \\
& \equiv \epsilon + (0^*10)^*00^* && \text{by (9.10)} \\
& \equiv \epsilon + (0^*10)^*0^*0 && \text{by (9.18)} \\
& \equiv \epsilon + (0 + 10)^*0 && \text{by (9.15).}
\end{aligned}
$$

(recursion)

naive method.

Extended regular expression (ERE) ← doesn't add power (Reg → DFA → operations → Reg)

$\emptyset \mid \varepsilon \mid a \mid r_1 + r_2 \mid r_1 \cdot r_2 \mid r^* \mid \underbrace{r_1 \cap r_2 \mid \sim r}_{\text{(extended re)}}$

"succintness" : descriptional complexity

sere (semi-extended re)

(ERE → RE : $n \mapsto 2^{2^{\cdots^{2}}}$ } height $\sim n$)