Pattern matching

What happens if, at a Unix/Linux shell prompt, you type

```
ls *
```

and press return?

Suppose the current directory contains files called `regfla.tex`, `regfla.aux`, `regfla.log`, `regfla.dvi`, and (strangely) `.aux`. What happens if you type

```
ls *.aux
```

and press return?

Alphabets

An ***alphabet*** is specified by giving a finite set, $\boldsymbol{\Sigma}$, whose elements are called ***symbols***. For us, any set qualifies as a possible alphabet, so long as it is finite.

Examples:

$\boldsymbol{\Sigma_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}}$ — $\mathbf{10}$-element set of decimal digits.

$\boldsymbol{\Sigma_2 = \{a, b, c, \ldots, x, y, z\}}$ — $\mathbf{26}$-element set of lower-case characters of the English language.

$\boldsymbol{\Sigma_3 = \{S \mid S \subseteq \Sigma_1\}}$ — $\mathbf{2^{10}}$-element set of all subsets of the alphabet of decimal digits.

Non-example:

$\mathbb{N} = \boldsymbol{\{0, 1, 2, 3, \ldots\}}$ — set of all non-negative whole numbers is not an alphabet, because it is infinite.

Strings over an alphabet

A ***string of length*** n (≥ 0) over an alphabet Σ is just an ordered n-tuple of elements of Σ, written without punctuation.

Example: if $\Sigma = \{a, b, c\}$, then a, ab, aac, and $bbac$ are strings over Σ of lengths one, two, three and four respectively.

$\Sigma^* \stackrel{\text{def}}{=}$ set of all strings over Σ of any finite length.

N.B. there is a unique string of length zero over Σ, called the ***null string*** (or ***empty string***) and denoted $\boxed{\varepsilon}$ (no matter which Σ we are talking about).

Concatenation of strings

The ***concatenation*** of two strings $u, v \in \Sigma^*$ is the string uv obtained by joining the strings end-to-end.

Examples: If $u = ab$, $v = ra$ and $w = cad$, then $vu = raab$, $uu = abab$ and $wv = cadra$.

This generalises to the concatenation of three or more strings.
E.g. $uvwuv = abracadabra$.

Regular expressions over an alphabet Σ

- each symbol $a \in \Sigma$ is a regular expression
- ε is a regular expression
- $\emptyset$ is a regular expression
- if r and s are regular expressions, then so is $(r|s)$
- if r and s are regular expressions, then so is rs
- if r is a regular expression, then so is $(r)^*$

Every regular expression is built up inductively, by *finitely many* applications of the above rules.

(N.B. we assume ε, $\emptyset$, $($, $)$, $|$, and $*$ are not symbols in Σ.)

Matching strings to regular expressions

- u matches $a \in \Sigma$ iff $u = a$
- u matches ε iff $u = \varepsilon$
- no string matches $\emptyset$
- u matches $r|s$ iff u matches either r or s
- u matches rs iff it can be expressed as the concatenation of two strings, $u = vw$, with v matching r and w matching s
- u matches r^* iff either $u = \varepsilon$, or u matches r, or u can be expressed as the concatenation of two or more strings, each of which matches r

Examples of matching, with $\Sigma = \{0, 1\}$

- $0|1$ is matched by each symbol in Σ
- $1(0|1)^*$ is matched by any string in Σ^* that starts with a '1'
- $((0|1)(0|1))^*$ is matched by any string of even length in Σ^*
- $(0|1)^*(0|1)^*$ is matched by any string in Σ^*
- $(\varepsilon|0)(\varepsilon|1)|11$ is matched by just the strings ε, 0, 1, 01, and 11
- $\emptyset 1|0$ is just matched by 0

Languages

A (formal) ***language*** L over an alphabet Σ is just a set of strings in Σ^*.

Thus any subset $L \subseteq \Sigma^*$ determines a language over Σ.

The ***language determined by a regular expression*** r over Σ is

$$L(r) \stackrel{\text{def}}{=} \{u \in \Sigma^* \mid u \text{ matches } r\}.$$

Two regular expressions r and s (over the same alphabet) are ***equivalent*** iff $L(r)$ and $L(s)$ are equal sets (i.e. have exactly the same members).

Some questions

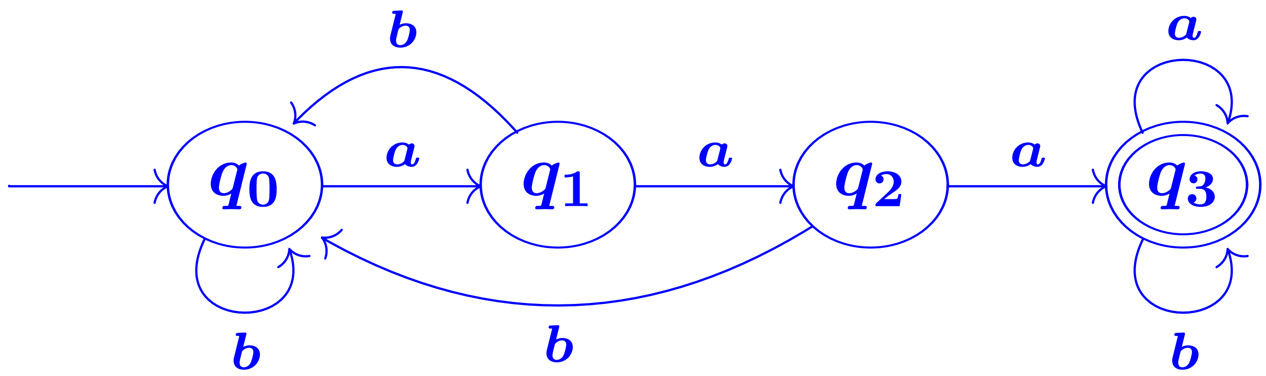
(a) Is there an algorithm which, given a string u and a regular expression r (over the same alphabet), computes whether or not u matches r?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s (over the same alphabet), computes whether or not they are equivalent? (Cf. Slide 8.)

(d) Is every language of the form $L(r)$?

Example of a finite automaton



States: q_0, q_1, q_2, q_3.

Input symbols: a, b.

Transitions: as indicated above.

Start state: q_0.

Accepting state(s): q_3.

$L(M)$, *language accepted* by a finite automaton M

consists of all strings u over its alphabet of input symbols satisfying $q_0 \xrightarrow{u}{}^{*} q$ with q_0 the start state and q some accepting state. Here

$$q_0 \xrightarrow{u}{}^{*} q$$

means, if $u = a_1 a_2 \ldots a_n$ say, that for some states $q_1, q_2, \ldots, q_n = q$ (not necessarily all distinct) there are transitions of the form

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} q_n = q.$$

N.B.

case $n = 0$: $\quad q \xrightarrow{\varepsilon}{}^{*} q' \quad$ iff $\quad q = q'$

case $n = 1$: $\quad q \xrightarrow{a}{}^{*} q' \quad$ iff $\quad q \xrightarrow{a} q'$.

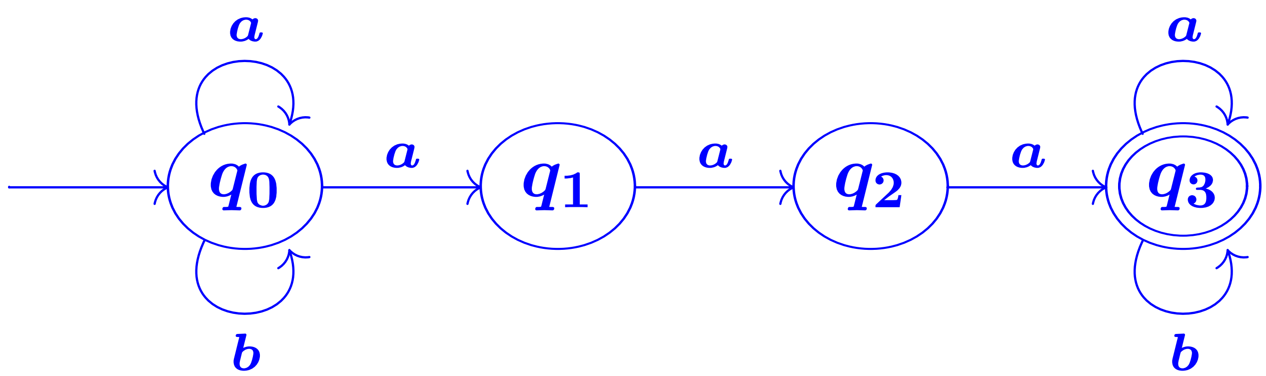
A ***non-deterministic finite automaton*** (NFA), $\boldsymbol{M}$,
is specified by

- a finite set $\boldsymbol{States_M}$ (of ***states***)
- a finite set $\boldsymbol{\Sigma_M}$ (the alphabet of ***input symbols***)
- for each $\boldsymbol{q \in States_M}$ and each $\boldsymbol{a \in \Sigma_M}$, a subset $\boldsymbol{\Delta_M(q,a) \subseteq States_M}$ (the set of states that can be reached from $\boldsymbol{q}$ with a single ***transition*** labelled $\boldsymbol{a}$)
- an element $\boldsymbol{s_M \in States_M}$ (the ***start state***)
- a subset $\boldsymbol{Accept_M \subseteq States_M}$ (of ***accepting states***)

Example of a non-deterministic finite automaton

Input alphabet: $\{a, b\}$.

States, transitions, start state, and accepting states as shown:



The language accepted by this automaton is the same as for the automaton on Slide 10, namely

$$\{u \in \{a, b\}^* \mid u \text{ contains three consecutive } a\text{'s}\}.$$

A ***deterministic finite automaton*** (DFA) is an NFA M with the property that for each $q \in States_M$ and $a \in \Sigma_M$, the finite set $\Delta_M(q, a)$ contains exactly one element—call it $\delta_M(q, a)$.

Thus in this case transitions in M are essentially specified by a ***next-state function***, δ_M, mapping each (state, input symbol)-pair (q, a) to the unique state $\delta_M(q, a)$ which can be reached from q by a transition labelled a:

$$q \xrightarrow{a} q' \quad \text{iff} \quad q' = \delta_M(q, a)$$

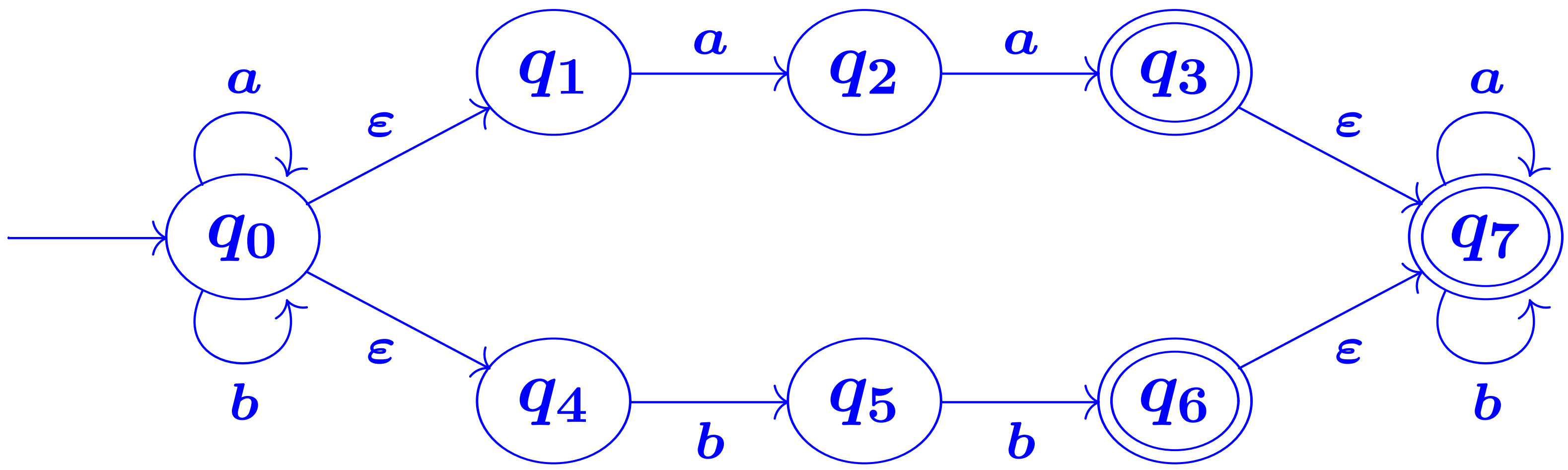
An ***NFA with ε-transitions*** (NFA$^\varepsilon$)

is specified by an NFA M together with a binary relation, called the ***ε-transition relation***, on the set $States_M$. We write

$$q \xrightarrow{\varepsilon} q'$$

to indicate that the pair of states (q, q') is in this relation.

Example (with input alphabet = $\{a, b\}$):



$L(M)$, *language accepted* by an NFA$^{\varepsilon}$ M

consists of all strings u over the alphabet Σ_M of input symbols satisfying $q_0 \stackrel{u}{\Rightarrow} q$ with q_0 the initial state and q some accepting state. Here $\cdot \stackrel{-}{\Rightarrow} \cdot$ is defined by:

$q \stackrel{\varepsilon}{\Rightarrow} q'$ iff $q = q'$ or there is a sequence $q \stackrel{\varepsilon}{\longrightarrow} \cdots q'$ of one or more ε-transitions in M from q to q'

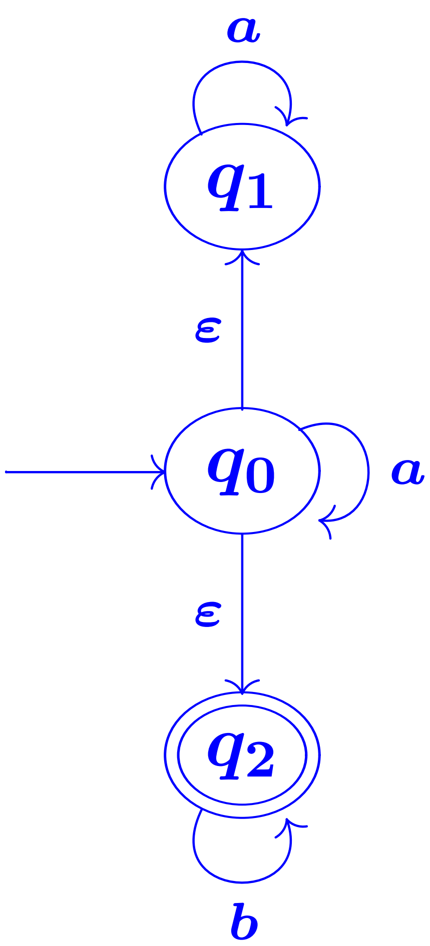
$q \stackrel{a}{\Rightarrow} q'$ (for $a \in \Sigma_M$) iff $q \stackrel{\varepsilon}{\Rightarrow} \cdot \stackrel{a}{\longrightarrow} \cdot \stackrel{\varepsilon}{\Rightarrow} q'$

$q \stackrel{ab}{\Rightarrow} q'$ (for $a, b \in \Sigma_M$) iff $q \stackrel{\varepsilon}{\Rightarrow} \cdot \stackrel{a}{\longrightarrow} \cdot \stackrel{\varepsilon}{\Rightarrow} \cdot \stackrel{b}{\longrightarrow} \cdot \stackrel{\varepsilon}{\Rightarrow} q'$

and similarly for longer strings

Example of the subset construction

M :



δ_{PM} :	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0, q_1, q_2\}$	$\{q_2\}$
$\{q_1\}$	$\{q_1\}$	$\emptyset$
$\{q_2\}$	$\emptyset$	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	$\{q_1\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_2\}$

Theorem. For each NFA$^{\varepsilon}$ M there is a DFA PM with the same alphabet of input symbols and accepting exactly the same strings as M, i.e. with $L(PM) = L(M)$

Definition of PM (refer to Slides 12 and 14):

- $States_{PM} \stackrel{\text{def}}{=} \{S \mid S \subseteq States_M\}$
- $\Sigma_{PM} \stackrel{\text{def}}{=} \Sigma_M$
- $S \xrightarrow{a} S'$ in PM iff $S' = \delta_{PM}(S, a)$, where $\delta_{PM}(S, a) \stackrel{\text{def}}{=} \{q' \mid \exists q \in S\ (q \stackrel{a}{\Rightarrow} q' \text{ in } M)\}$
- $s_{PM} \stackrel{\text{def}}{=} \{q \mid s_M \stackrel{\varepsilon}{\Rightarrow} q\}$
- $Accept_{PM} \stackrel{\text{def}}{=}$
  $\{S \in States_{PM} \mid \exists q \in S\ (q \in Accept_M)\}$

Definition

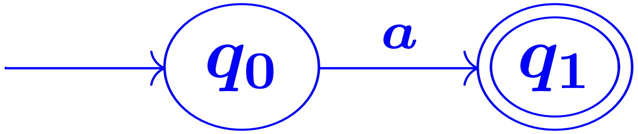
A language is ***regular*** iff it is the set of strings accepted by some deterministic finite automaton.

Kleene's Theorem

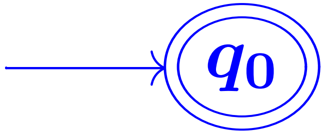
(a) *For any regular expression* r, $L(r)$ *is a regular language* (cf. Slide 8).

(b) *Conversely, every regular language is the form* $L(r)$ *for some regular expression* r.

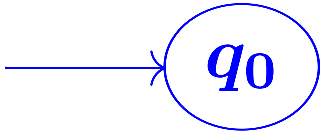
NFAs for atomic regular expressions



just accepts the one-symbol string a

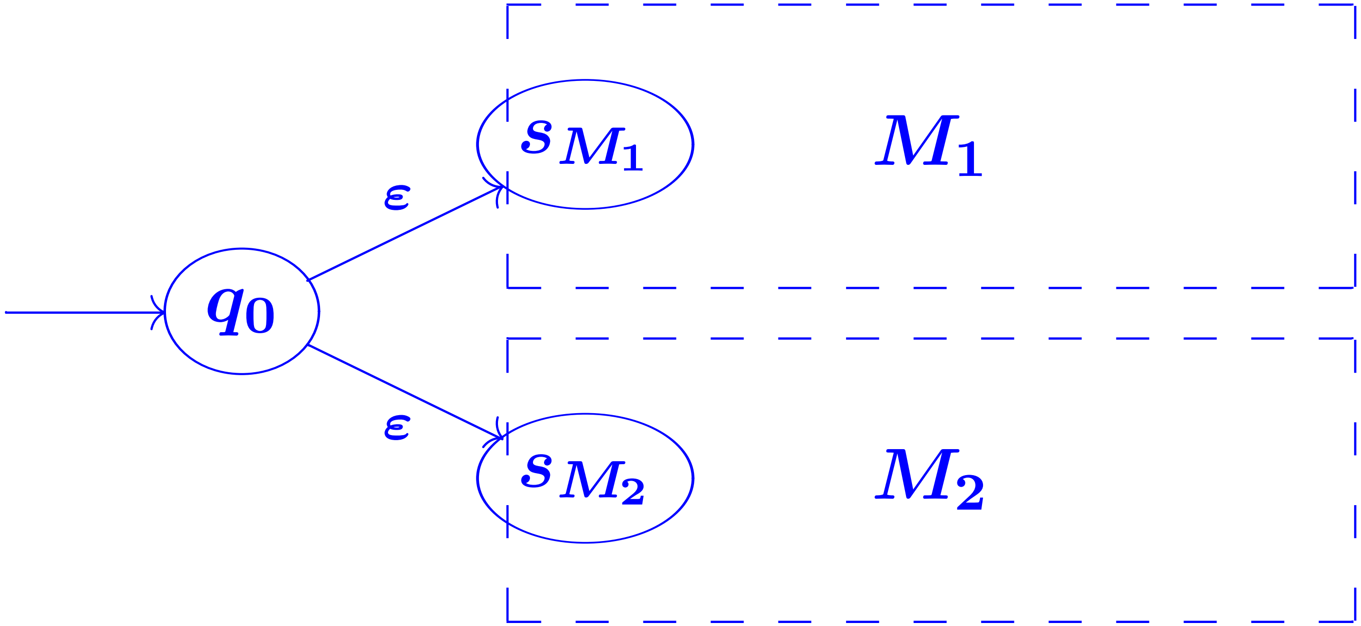


just accepts the null string, ε



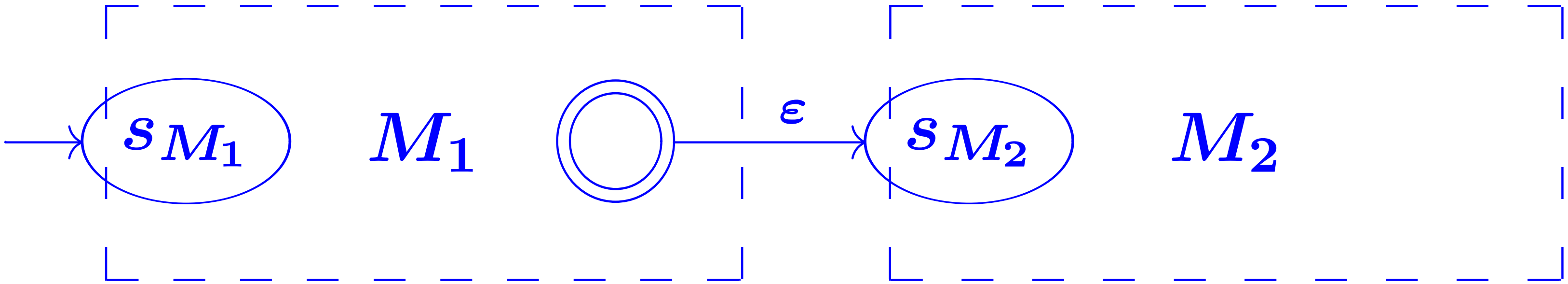
accepts no strings

$Union(M_1, M_2)$



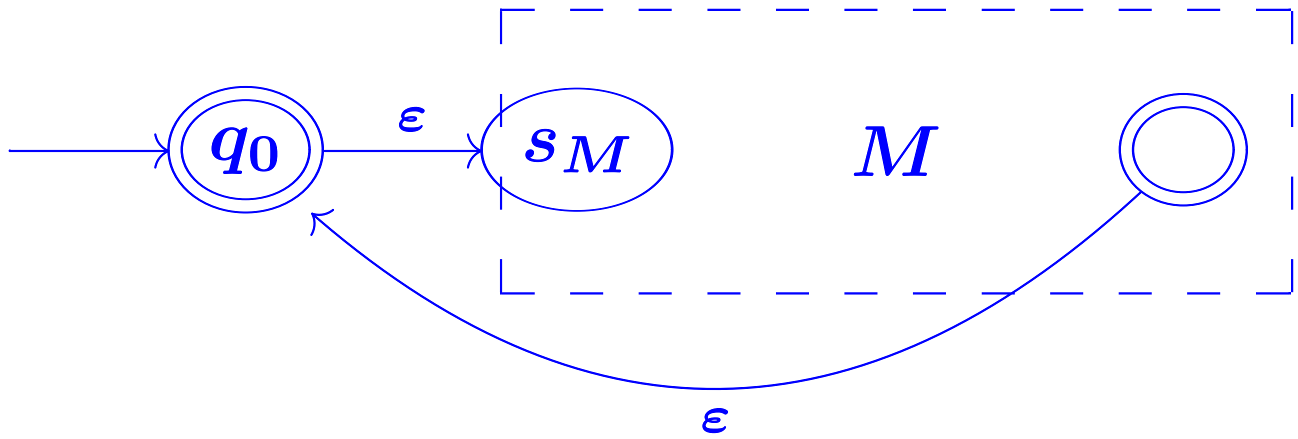
Set of accepting states is union of $Accept_{M_1}$ and $Accept_{M_2}$.

$Concat(M_1, M_2)$



Set of accepting states is $Accept_{M_2}$.

$Star(M)$



The only accepting state of $Star(M)$ is q_0.

Lemma *Given an NFA* M*, for each subset* $Q \subseteq \mathit{States}_M$ *and each pair of states* $q, q' \in \mathit{States}_M$*, there is a regular expression* $r^Q_{q,q'}$ *satisfying*

$$L(r^Q_{q,q'}) = \{u \in (\Sigma_M)^* \mid q \xrightarrow{u}{}^* q' \text{ in } M \text{ with all intermediate states of the sequence in } Q\}.$$

Hence $L(M) = L(r)$, where $r = r_1 | \cdots | r_k$ and

$k =$ number of accepting states,

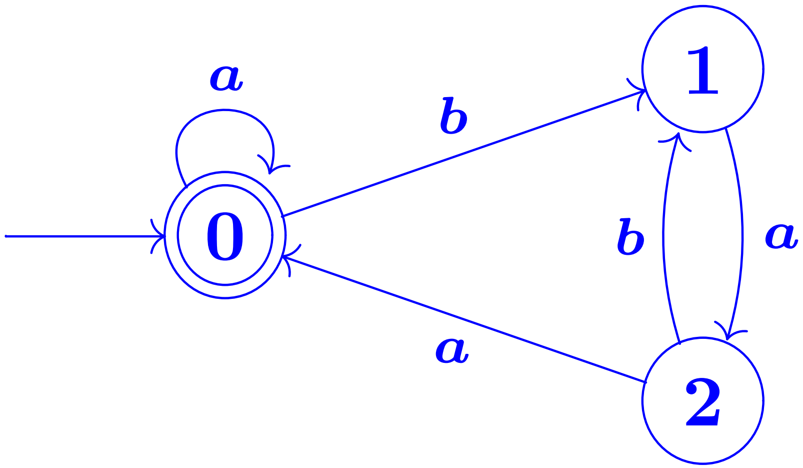
$r_i = r^Q_{s,q_i}$ with $Q = \mathit{States}_M$,

$s =$ start state,

$q_i = i$th accepting state.

(In case $k = 0$, take r to be the regular expression $\emptyset$.)

Example



Direct inspection yields:

$r_{i,j}^{\{0\}}$	0	1	2
0			
1	$\emptyset$	ε	a
2	aa^*	a^*b	ε

$r_{i,j}^{\{0,2\}}$	0	1	2
0	a^*	a^*b	
1			
2			

$Not(M)$

- $States_{Not(M)} \stackrel{\text{def}}{=} States_M$
- $\Sigma_{Not(M)} \stackrel{\text{def}}{=} \Sigma_M$
- transitions of $Not(M)$ = transitions of M
- start state of $Not(M)$ = start state of M
- $Accept_{Not(M)} = \{q \in States_M \mid q \notin Accept_M\}$.

Provided M is a *deterministic* finite automaton, then u is accepted by $Not(M)$ iff it is not accepted by M:

$$L(Not(M)) = \{u \in \Sigma^* \mid u \notin L(M)\}.$$

$And(M_1, M_2)$

- states of $And(M_1, M_2)$ are all ordered pairs (q_1, q_2) with $q_1 \in States_{M_1}$ and $q_2 \in States_{M_2}$
- alphabet of $And(M_1, M_2)$ is the common alphabet of M_1 and M_2
- $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ in $And(M_1, M_2)$ iff $q_1 \xrightarrow{a} q_1'$ in M_1 and $q_2 \xrightarrow{a} q_2'$ in M_2
- start state of $And(M_1, M_2)$ is (s_{M_1}, s_{M_2})
- (q_1, q_2) accepting in $And(M_1, M_2)$ iff q_1 accepting in M_1 and q_2 accepting in M_2.

Examples of non-regular languages

- The set of strings over $\{(,), a, b, \ldots, z\}$ in which the parentheses '$($' and '$)$' occur well-nested.
- The set of strings over $\{a, b, \ldots, z\}$ which are palindromes, i.e. which read the same backwards as forwards.
- $\{a^n b^n \mid n \geq 0\}$

The Pumping Lemma

For every regular language L, there is a number $\ell \geq 1$ satisfying the ***pumping lemma property***:

all $w \in L$ with $length(w) \geq \ell$ can be expressed as a concatenation of three strings, $w = u_1 v u_2$, where u_1, v and u_2 satisfy:

- $length(v) \geq 1$
  (i.e. $v \neq \varepsilon$)
- $length(u_1 v) \leq \ell$
- for all $n \geq 0$, $u_1 v^n u_2 \in L$
  (i.e. $u_1 u_2 \in L$, $u_1 v u_2 \in L$ [but we knew that anyway], $u_1 v v u_2 \in L$, $u_1 v v v u_2 \in L$, etc).

If $n \geq \ell =$ number of states of M, then in

$$s_M = \underbrace{q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_\ell} q_\ell}_{\ell+1 \text{ states}} \cdots \xrightarrow{a_n} q_n \in Accept_M$$

$q_0, \ldots, q_\ell$ can't all be distinct states. So $q_i = q_j$ for some $0 \leq i < j \leq \ell$. So the above transition sequence looks like

$$s_M = q_0 \xrightarrow{u_1}{}^{*} \; q_i \; = \; q_j \overset{v}{\curvearrowleft}{}^{*} \; \xrightarrow{u_2}{}^{*} q_n \in Accept_M$$

where

$$u_1 \stackrel{\text{def}}{=} a_1 \ldots a_i \qquad v \stackrel{\text{def}}{=} a_{i+1} \ldots a_j \qquad u_2 \stackrel{\text{def}}{=} a_{j+1} \ldots a_n.$$

How to use the Pumping Lemma to prove that a language L is *not* regular

For each $\ell \geq 1$, find some $w \in L$ of length $\geq \ell$ so that

$$(\dagger) \quad \left\{ \begin{array}{l} \text{no matter how } w \text{ is split into three, } w = u_1 v u_2, \\ \text{with } length(u_1 v) \leq \ell \text{ and } length(v) \geq 1, \\ \\ \text{there is some } n \geq 0 \text{ for which } u_1 v^n u_2 \text{ is } \textit{not} \text{ in } L. \end{array} \right.$$

Examples

(i) $L_1 \stackrel{\text{def}}{=} \{a^n b^n \mid n \geq 0\}$ *is not regular.*

[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†) on Slide 31.]

(ii) $L_2 \stackrel{\text{def}}{=} \{w \in \{a,b\}^* \mid w \text{ a palindrome}\}$ *is not regular.*

[For each $\ell \geq 1$, $a^\ell b a^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(iii) $L_3 \stackrel{\text{def}}{=} \{a^p \mid p \text{ prime}\}$ *is not regular.*

[For each $\ell \geq 1$, we can find a prime p with $p > 2\ell$ and then $a^p \in L_3$ has length $\geq \ell$ and has property (†).]

Example of a non-regular language that satisfies the ‘pumping lemma property’

$$L \stackrel{\text{def}}{=} \{c^m a^n b^n \mid m \geq 1 \text{ and } n \geq 0\}$$
$$\cup$$
$$\{a^m b^n \mid m, n \geq 0\}$$

satisfies the pumping lemma property on Slide 29 with $\ell = 1$.

[For any $w \in L$ of length ≥ 1, can take $u_1 = \varepsilon$, $v =$ first letter of w, $u_2 =$ rest of w.]

But L is not regular. [See Exercise **??**.]

Lemma *If a DFA M accepts any string at all, it accepts one whose length is less than the number of states in M.*

Proof. Suppose M has ℓ states (so $\ell \geq 1$). If $L(M)$ is not empty, then we can find an element of it of shortest length, $a_1 a_2 \ldots a_n$ say (where $n \geq 0$). Thus there is a transition sequence

$$s_M = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_n} q_n \in Accept_M.$$

If $n \geq \ell$, then not all the $n + 1$ states in this sequence can be distinct and we can shorten it as on Slide 30. But then we would obtain a strictly shorter string in $L(M)$ contradicting the choice of $a_1 a_2 \ldots a_n$. So we must have $n < \ell$. □

Some production rules for ‘English’ sentences

$$
\begin{aligned}
\texttt{SENTENCE} &\longrightarrow \texttt{SUBJECT VERB OBJECT} \\
\texttt{SUBJECT} &\longrightarrow \texttt{ARTICLE NOUNPHRASE} \\
\texttt{OBJECT} &\longrightarrow \texttt{ARTICLE NOUNPHRASE} \\
\texttt{ARTICLE} &\longrightarrow \texttt{a} \\
\texttt{ARTICLE} &\longrightarrow \texttt{the} \\
\texttt{NOUNPHRASE} &\longrightarrow \texttt{NOUN} \\
\texttt{NOUNPHRASE} &\longrightarrow \texttt{ADJECTIVE NOUN} \\
\texttt{ADJECTIVE} &\longrightarrow \texttt{big} \\
\texttt{ADJECTIVE} &\longrightarrow \texttt{small} \\
\texttt{NOUN} &\longrightarrow \texttt{cat} \\
\texttt{NOUN} &\longrightarrow \texttt{dog} \\
\texttt{VERB} &\longrightarrow \texttt{eats}
\end{aligned}
$$

A derivation

$$
\begin{aligned}
\underline{\texttt{SENTENCE}} &\longrightarrow \underline{\texttt{SUBJECT}}\ \texttt{VERB OBJECT} \\
&\longrightarrow \underline{\texttt{ARTICLE}}\ \texttt{NOUNPHRASE VERB OBJECT} \\
&\longrightarrow \texttt{the NOUNPHRASE}\ \underline{\texttt{VERB}}\ \texttt{OBJECT} \\
&\longrightarrow \texttt{the}\ \underline{\texttt{NOUNPHRASE}}\ \texttt{eats OBJECT} \\
&\longrightarrow \texttt{the}\ \underline{\texttt{ADJECTIVE}}\ \texttt{NOUN eats OBJECT} \\
&\longrightarrow \texttt{the big}\ \underline{\texttt{NOUN}}\ \texttt{eats OBJECT} \\
&\longrightarrow \texttt{the big cat eats}\ \underline{\texttt{OBJECT}} \\
&\longrightarrow \texttt{the big cat eats}\ \underline{\texttt{ARTICLE}}\ \texttt{NOUNPHRASE} \\
&\longrightarrow \texttt{the big cat eats a}\ \underline{\texttt{NOUNPHRASE}} \\
&\longrightarrow \texttt{the big cat eats a}\ \underline{\texttt{ADJECTIVE}}\ \texttt{NOUN} \\
&\longrightarrow \texttt{the big cat eats a small}\ \underline{\texttt{NOUN}} \\
&\longrightarrow \texttt{the big cat eats a small dog}
\end{aligned}
$$

Example of Backus-Naur Form (BNF)

Terminals:

$$\mathtt{x} \quad ' \quad + \quad - \quad * \quad (\quad)$$

Non-terminals:

$$\mathtt{id} \quad \mathtt{op} \quad \mathtt{exp}$$

Start symbol:

$$\mathtt{exp}$$

Productions:

$$\begin{array}{rcl} \mathtt{id} & ::= & \mathtt{x} \mid \mathtt{id}' \\ \mathtt{op} & ::= & + \mid - \mid * \\ \mathtt{exp} & ::= & \mathtt{id} \mid \mathtt{exp}\ \mathtt{op}\ \mathtt{exp} \mid (\mathtt{exp}) \end{array}$$

A context-free grammar for the language $\{a^n b^n \mid n \geq 0\}$

Terminals:

$$a \qquad b$$

Non-terminal:

$$I$$

Start symbol:

$$I$$

Productions:

$$I ::= \varepsilon \mid aIb$$

Every regular language is context-free

Given a DFA M, the set $L(M)$ of strings accepted by M can be generated by the following context-free grammar:

set of terminals = Σ_M

set of non-terminals = $States_M$

start symbol = start state of M

productions of two kinds:

$q \rightarrow aq'$	whenever $q \xrightarrow{a} q'$ in M
$q \rightarrow \varepsilon$	whenever $q \in Accept_M$

Definition A context-free grammar is ***regular*** iff all its productions are of the form

$$x \longrightarrow uy$$

or

$$x \longrightarrow u$$

where u is a string of terminals and x and y are non-terminals.

Theorem

(a) *Every language generated by a regular grammar is a regular language (i.e. is the set of strings accepted by some DFA).*

(b) *Every regular language can be generated by a regular grammar.*

Example of the construction used in the proof of the Theorem on Slide 40

regular grammar:

$$S \to abX$$
$$X \to bbY$$
$$Y \to X$$
$$X \to a$$
$$Y \to \varepsilon$$

(start symbol = S)

$\rightsquigarrow$ NFA$^{\varepsilon}$: