Formal Languages and Automata

5 lectures for
University of Cambridge
2018–2019 Computer Science Tripos
Part IA Discrete Mathematics
by Prof. Frank Stajano

Originally written by **Prof. Andrew Pitts**

Minor tweaks by Prof. Ian Leslie and Prof. Frank Stajano

Revision 12 of 2018-12-31 14:12:10 +0000 (Mon, 31 Dec 2018)

Syllabus for this part of the course

- Inductive definitions using rules and proofs by rule induction.
- Abstract syntax trees.
- Regular expressions and pattern matching.
- Finite automata and regular languages: Kleene's theorem.
- The Pumping Lemma.

Common theme: mathematical techniques for defining formal languages and reasoning about their properties.

Key concepts: inductive definitions, automata

Relevant to:

Part IB Compiler Construction, Computation Theory, Complexity Theory, Semantics of Programming Languages

Part II Natural Language Processing, Optimising Compilers, Denotational Semantics, Temporal Logic and Model Checking

N.B. we do not cover the important topic of context-free grammars, which prior to 2013/14 was part of the CST IA course *Regular Languages and Finite Automata* that has been subsumed into this course.

Formal Languages

Alphabets

An **alphabet** is specified by giving a finite set, Σ, whose elements are called **symbols**. For us, any set qualifies as a possible alphabet, so long as it is finite.

Examples:

- $\{0,1,2,3,4,5,6,7,8,9\}$, **10**-element set of decimal digits.
- $\{a,b,c,\ldots,x,y,z\}$, **26**-element set of lower-case characters of the English language.
- $\{S \mid S \subseteq \{0,1,2,3,4,5,6,7,8,9\}\}$, $\mathbf{2^{10}}$-element set of all subsets of the alphabet of decimal digits.

Non-example:

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

Strings over an alphabet

A **string of length** n (for $n = 0, 1, 2, \ldots$) over an alphabet Σ is just an ordered n-tuple of elements of Σ, written without punctuation.

Σ^* denotes set of all strings over Σ of any finite length.

Examples:

notation for the string of length 0

- If $\Sigma = \{a, b, c\}$, then ε, a, ab, aac, and $bbac$ are strings over Σ of lengths zero, one, two, three and four respectively.
- If $\Sigma = \{a\}$, then Σ^* contains ε, a, aa, aaa, $aaaa$, etc.

In general, a^n denotes the string of length n just containing a symbols

Strings over an alphabet

A **string of length** n (for $n = 0, 1, 2, \ldots$) over an alphabet Σ is just an ordered n-tuple of elements of Σ, written without punctuation.

Σ^* denotes set of all strings over Σ of any finite length.

Examples:

- If $\Sigma = \{a, b, c\}$, then ε, a, ab, aac, and $bbac$ are strings over Σ of lengths zero, one, two, three and four respectively.
- If $\Sigma = \{a\}$, then Σ^* contains ε, a, aa, aaa, $aaaa$, etc.
- If $\Sigma = \emptyset$ (the empty set), then what is Σ^*?

Strings over an alphabet

A **string of length** n (for $n = 0, 1, 2, \ldots$) over an alphabet Σ is just an ordered n-tuple of elements of Σ, written without punctuation.

Σ^* denotes set of all strings over Σ of any finite length.

Examples:

- If $\Sigma = \{a, b, c\}$, then ε, a, ab, aac, and $bbac$ are strings over Σ of lengths zero, one, two, three and four respectively.
- If $\Sigma = \{a\}$, then Σ^* contains ε, a, aa, aaa, $aaaa$, etc.
- If $\Sigma = \emptyset$ (the empty set), then $\Sigma^* = \{\varepsilon\}$.

Concatenation of strings

The **concatenation** of two strings u and v is the string uv obtained by joining the strings end-to-end. This generalises to the concatenation of three or more strings.

Examples:

If $\Sigma = \{a, b, c, \ldots, z\}$ and $u, v, w \in \Sigma^*$ are $u = ab$, $v = ra$ and $w = cad$, then

$$vu = raab$$
$$uu = abab$$
$$wv = cadra$$
$$uvwuv = abracadabra$$

Formal languages

An extensional view of what constitutes a formal language is that it is completely determined by the set of 'words in the dictionary':

Given an alphabet Σ, we call any subset of Σ^* a (formal) **language** over the alphabet Σ.

We will use inductive definitions to describe languages in terms of grammatical rules for generating subsets of Σ^*.

Inductive Definitions

Axioms and rules

for inductively defining a subset of a given set U

- **axioms** $\dfrac{}{a}$ are specified by giving an element a of U
- **rules** $\dfrac{h_1 \; h_2 \; \cdots \; h_n}{c}$

  are specified by giving a finite subset $\{h_1, h_2, \ldots, h_n\}$ of U (the **hypotheses** of the rule) and an element c of U (the **conclusion** of the rule)

Derivations

Given a set of axioms and rules for inductively defining a subset of a given set U, a **derivation** (or proof) that a particular element $u \in U$ is in the subset is by definition

> a finite rooted tree with vertexes labelled by elements of U and such that:
>
> - the root of the tree is u (the conclusion of the whole derivation),
> - each vertex of the tree is the conclusion of a rule whose hypotheses are the children of the node,
> - each leaf of the tree is an axiom.

Example

$U = \{a, b\}^*$

axiom: $\dfrac{}{\varepsilon}$

rules: $\dfrac{u}{aub}$ $\dfrac{u}{bua}$ $\dfrac{u \quad v}{uv}$ (for all $u, v \in U$)

Example derivations:

$$\dfrac{\dfrac{\varepsilon}{ab} \quad \dfrac{\dfrac{\varepsilon}{ab}}{aabb}}{abaabb} \qquad \dfrac{\dfrac{\dfrac{\varepsilon}{ba} \quad \dfrac{\varepsilon}{ab}}{baab}}{abaabb}$$

Inductively defined subsets

Given a set of axioms and rules over a set U, the subset of U **inductively defined** by the axioms and rules consists of all and only the elements $u \in U$ for which there is a derivation with conclusion u.

For example, for the axioms and rules on Slide 15

- $abaabb$ is in the subset they inductively define (as witnessed by either derivation on that slide)
- $abaab$ is not in that subset (there is no derivation with that conclusion – why?)

(In fact $u \in \{a,b\}^*$ is in the subset iff it contains the same number of a and b symbols.)

Example: transitive closure

Given a binary relation $R \subseteq X \times X$ on a set X, its **transitive closure** R^+ is the smallest (for subset inclusion) binary relation on X which contains R and which is **transitive** ($\forall x, y, z \in X.\ (x, y) \in R^+ \ \&\ (y, z) \in R^+ \Rightarrow (x, z) \in R^+$).

R^+ is equal to the subset of $X \times X$ inductively defined by

axioms $\dfrac{}{(x, y)}$ (for all $(x, y) \in R$)

rules $\dfrac{(x, y) \qquad (y, z)}{(x, z)}$ (for all $x, y, z \in X$)

Example: reflexive-transitive closure

Given a binary relation $R \subseteq X \times X$ on a set X, its **reflexive-transitive closure** R^* is defined to be the smallest binary relation on X which contains R, is both transitive and **reflexive** ($\forall x \in X.\ (x,x) \in R^*$).

R^* is equal to the subset of $X \times X$ inductively defined by

axioms $\dfrac{}{(x,y)}$ (for all $(x,y) \in R$) $\qquad \dfrac{}{(x,x)}$ (for all $x \in X$)

rules $\dfrac{(x,y) \qquad (y,z)}{(x,z)}$ (for all $x,y,z \in X$)

Example: reflexive-transitive closure

Given a binary relation $R \subseteq X \times X$ on a set X, its **reflexive-transitive closure** R^* is defined to be the smallest binary relation on X which contains R, is both transitive and **reflexive** ($\forall x \in X.\ (x,x) \in R^*$).

R^* is equal to the subset of $X \times X$ inductively defined by

axioms $\dfrac{}{(x,y)}$ (for all $(x,y) \in R$) $\qquad \dfrac{}{(x,x)}$ (for all $x \in X$)

rules $\dfrac{(x,y) \qquad (y,z)}{(x,z)}$ (for all $x,y,z \in X$)

we can use Rule Induction (Slide 20) to prove this

Rule Induction

Theorem. The subset $I \subseteq U$ inductively defined by a collection of axioms and rules is closed under them and is the least such subset: if $S \subseteq U$ is also closed under the axioms and rules, then $I \subseteq S$.

Given axioms and rules for inductively defining a subset of a set U, we say that a subset $S \subseteq U$ is **closed under the axioms and rules** if

- for every axiom $\dfrac{}{a}$, it is the case that $a \in S$
- for every rule $\dfrac{h_1 \; h_2 \; \cdots \; h_n}{c}$, if $h_1, h_2, \ldots, h_n \in S$, then $c \in S$.

Rule Induction

Theorem. The subset $I \subseteq U$ inductively defined by a collection of axioms and rules is closed under them and is the least such subset: if $S \subseteq U$ is also closed under the axioms and rules, then $I \subseteq S$.

We use the theorem as method of proof: given a property $P(u)$ of elements of U, to prove $\forall u \in I.\, P(u)$ it suffices to show

- **base cases:** $P(a)$ holds for each axiom $\dfrac{}{a}$
- **induction steps:** $P(h_1)\ \&\ P(h_2)\ \&\ \cdots\ \&\ P(h_n) \Rightarrow P(c)$ holds for each rule $\dfrac{h_1\ h_2\ \cdots\ h_n}{c}$

(To see this, apply the theorem with $S = \{u \in U \mid P(u)\}$.)

Example: reflexive-transitive closure

Given a binary relation $R \subseteq X \times X$ on a set X, its **reflexive-transitive closure** R^* is defined to be the smallest binary relation on X which contains R, is both transitive and **reflexive** ($\forall x \in X.\ (x,x) \in R^*$).

R^* is equal to the subset of $X \times X$ inductively defined by

axioms $\dfrac{}{(x,y)}$ (for all $(x,y) \in R$) $\qquad \dfrac{}{(x,x)}$ (for all $x \in X$)

rules $\dfrac{(x,y) \qquad (y,z)}{(x,z)}$ (for all $x,y,z \in X$)

we can use Rule Induction (Slide 20) to prove this, since $S \subseteq X \times X$ being closed under the axioms & rules is the same as it containing R, being reflexive and being transitive.

Example using rule induction

Let I be the subset of $\{a,b\}^*$ inductively defined by the axioms and rules on Slide 15.

For $u \in \{a,b\}^*$, let $P(u)$ be the property

u contains the same number of a and b symbols

We can prove $\forall u \in I.\, P(u)$ by rule induction:

- **base case:** $P(\varepsilon)$ is true (the number of as and bs is zero!)

Example using rule induction

Let I be the subset of $\{a,b\}^*$ inductively defined by the axioms and rules on Slide 15.

For $u \in \{a,b\}^*$, let $P(u)$ be the property

u contains the same number of a and b symbols

We can prove $\forall u \in I.\, P(u)$ by rule induction:

- **base case:** $P(\varepsilon)$ is true (the number of as and bs is zero!)
- **induction steps:** if $P(u)$ and $P(v)$ hold, then clearly so do $P(aub)$, $P(bua)$ and $P(uv)$.

(It's not so easy to show $\forall u \in \{a,b\}^*.\, P(u) \Rightarrow u \in I$ – rule induction for I is not much help for that.)

Abstract Syntax Trees

Concrete syntax: strings of symbols

- possibly including symbols to disambiguate the semantics (brackets, white space, *etc*),
- or that have no semantic content (*e.g.* syntax for comments).

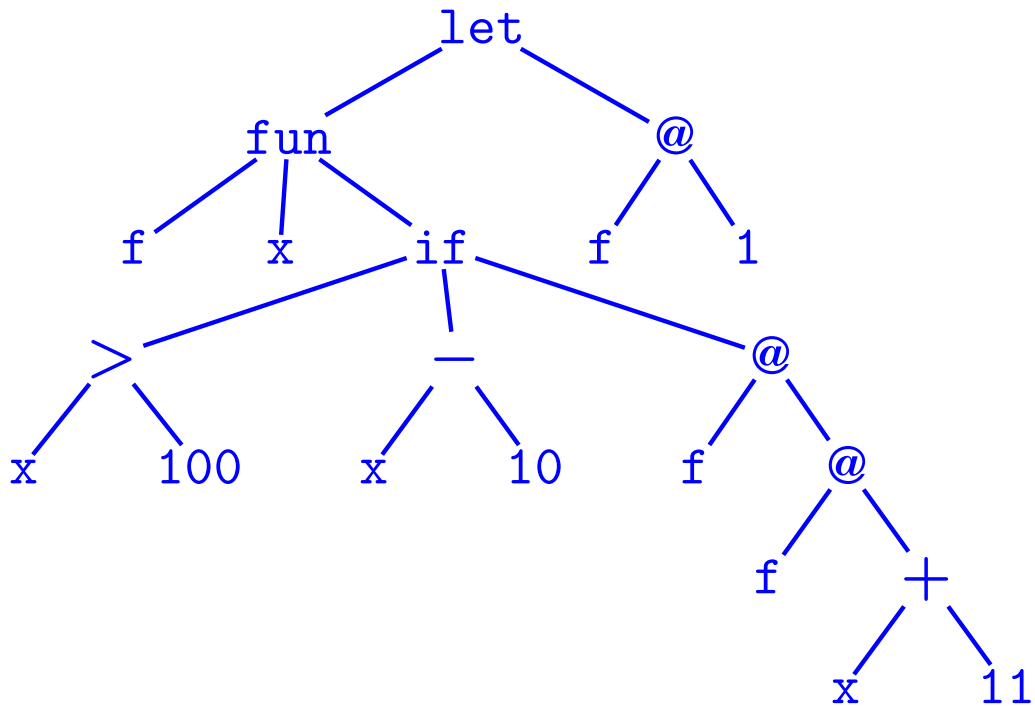
For example, an ML expression:

```
let fun f x =
  if x > 100 then x - 10
  else f ( f ( x + 11 ) )
in f 1 end
(* v a l u e   i s   9 9 *)
```

Abstract syntax: finite rooted trees

- vertexes with n children are labelled by operators expecting n arguments (n-**ary** operators) – in particular leaves are labelled with $\mathbf{0}$-ary (nullary) operators (constants, variables, *etc*)
- label of the root gives the 'outermost form' of the whole phrase

E.g. for the ML expression on Slide 25:



Regular expressions (concrete syntax)

over a given alphabet Σ.

Let Σ' be the 6-element set $\{\epsilon, \emptyset, |, ^*, (,)\}$ (assumed disjoint from Σ)

$$U = (\Sigma \cup \Sigma')^*$$

axioms: $\dfrac{}{a}$ $\qquad$ $\dfrac{}{\epsilon}$ $\qquad$ $\dfrac{}{\emptyset}$

rules: $\dfrac{r}{(r)}$ $\qquad$ $\dfrac{r \quad s}{r|s}$ $\qquad$ $\dfrac{r \quad s}{rs}$ $\qquad$ $\dfrac{r}{r^*}$

(where $a \in \Sigma$ and $r, s \in U$)

Some derivations of regular expressions (assuming $a, b \in \Sigma$)

$$\dfrac{\epsilon \qquad \dfrac{a \qquad \dfrac{b}{b^*}}{ab^*}}{\epsilon|ab^*}$$

$$\dfrac{\dfrac{\epsilon \qquad a}{\epsilon|a} \qquad \dfrac{b}{b^*}}{\epsilon|ab^*}$$

$$\dfrac{\epsilon \qquad \dfrac{\dfrac{a \qquad b}{ab}}{ab^*}}{\epsilon|ab^*}$$

$$\dfrac{\epsilon \qquad \dfrac{a \qquad \dfrac{\dfrac{b}{b^*}}{(b^*)}}{a(b^*)}}{\epsilon|(a(b^*))}$$

$$\dfrac{\dfrac{\epsilon \qquad a}{\epsilon|a} \qquad \dfrac{b}{b^*}}{(\epsilon|a)(b^*)}$$

$$\dfrac{\epsilon \qquad \dfrac{\dfrac{\dfrac{a \qquad b}{ab}}{(ab)}}{(ab)^*}}{\epsilon|((ab)^*)}$$

Regular expressions (abstract syntax)

The 'signature' for regular expression abstract syntax trees (over an alphabet Σ) consists of

- binary operators Union and Concat
- unary operator Star
- nullary operators (constants) Null, Empty and Sym_a (one for each $a \in \Sigma$).

Regular expressions (abstract syntax)

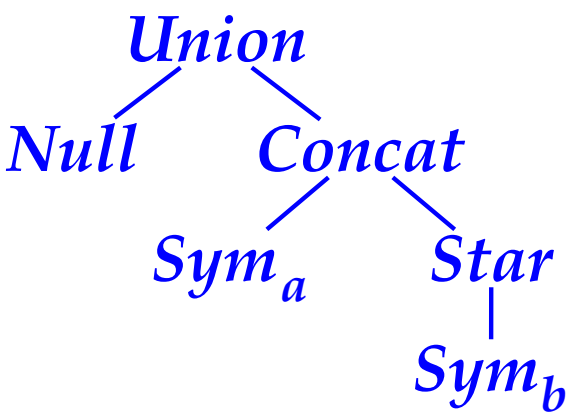
The 'signature' for regular expression abstract syntax trees (over an alphabet Σ) as an ML datatype declaration:

```
datatype 'a RE = Union of ('a RE) * ('a RE)
               | Concat of ('a RE) * ('a RE)
               | Star of 'a RE
               | Null
               | Empty
               | Sym of 'a
```

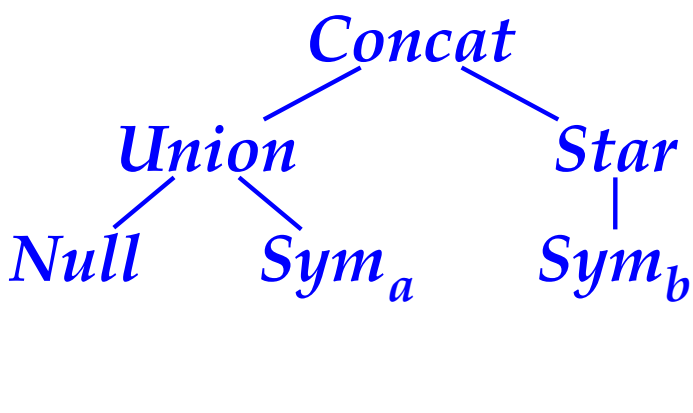
(the type `'a RE` is parameterised by a type variable `'a` standing for the alphabet Σ)

Some abstract syntax trees of regular expressions (assuming $a, b \in \Sigma$)

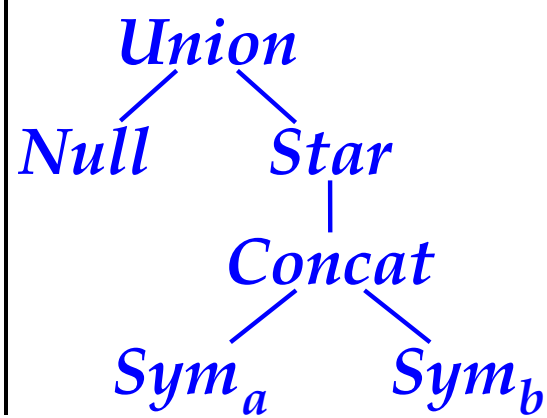
1.



2.



3.



(*cf.* examples on Slide 28)

We will use a textual representation of trees, for example:

1. $Union(Null, Concat(Sym_a, Star(Sym_b)))$
2. $Concat(Union(Null, Sym_a), Star(Sym_b))$
3. $Union(Null, Star(Concat(Sym_a, Sym_b)))$

Relating concrete and abstract syntax

for regular expressions over an alphabet Σ, via an inductively defined relation $\sim$ between strings and trees:

$$\frac{}{a \sim Sym_a} \qquad \frac{}{\epsilon \sim Null} \qquad \frac{}{\emptyset \sim Empty}$$

$$\frac{r \sim R}{(r) \sim R} \qquad \frac{r \sim R \qquad s \sim S}{r|s \sim Union(R,S)}$$

$$\frac{r \sim R \qquad s \sim S}{rs \sim Concat(R,S)} \qquad \frac{r \sim R}{r^* \sim Star(R)}$$

For example:

$$\epsilon|(a(b^*)) \sim Union(Null, Concat(Sym_a, Star(Sym_b)))$$
$$\epsilon|ab^* \sim Union(Null, Concat(Sym_a, Star(Sym_b)))$$
$$\epsilon|ab^* \sim Concat(Union(Null, Sym_a), Star(Sym_b))$$

Thus $\sim$ is a 'many-many' relation between strings and trees.

- **Parsing:** algorithms for producing abstract syntax trees $parse(r)$ from concrete syntax r, satisfying $r \sim parse(r)$.
- **Pretty printing:** algorithms for producing concrete syntax $pp(R)$ from abstract syntax trees R, satisfying $pp(R) \sim R$.

(See CST IB Compiler construction course.)

Matching

Each regular expression r over an alphabet Σ determines a language $L(r) \subseteq \Sigma^*$. The strings u in $L(r)$ are by definition the ones that **match** r, where

- u matches the regular expression a (where $a \in \Sigma$) iff $u = a$
- u matches the regular expression ϵ iff u is the null string ε
- no string matches the regular expression $\emptyset$
- u matches $r|s$ iff it either matches r, or it matches 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.

Inductive definition of matching

$U = \Sigma^* \times \{\text{regular expressions over } \Sigma\}$

abstract syntax trees

axioms: $\dfrac{}{(a,a)}$ $\dfrac{}{(\varepsilon,\epsilon)}$ $\dfrac{}{(\varepsilon,r^*)}$

rules:

$$\frac{(u,r)}{(u,r|s)} \qquad \frac{(u,s)}{(u,r|s)}$$

$$\frac{(v,r) \quad (w,s)}{(vw,rs)} \qquad \frac{(u,r) \quad (v,r^*)}{(uv,r^*)}$$

(No axiom/rule involves the empty regular expression $\emptyset$ – why?)

Examples of matching

Assuming $\Sigma = \{a, b\}$, then:

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

Some questions

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

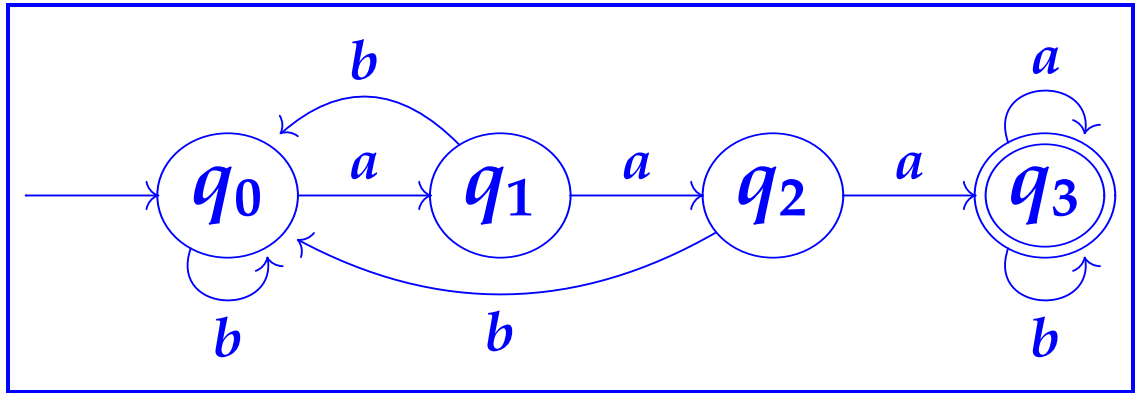
(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

Finite Automata

Example of a finite automaton

$M \triangleq$ 

- set of states: $\{q_0, q_1, q_2, q_3\}$
- input alphabet: $\{a, b\}$
- transitions, labelled by input symbols: as indicated by the above directed graph
- start state: q_0
- accepting state(s): q_3

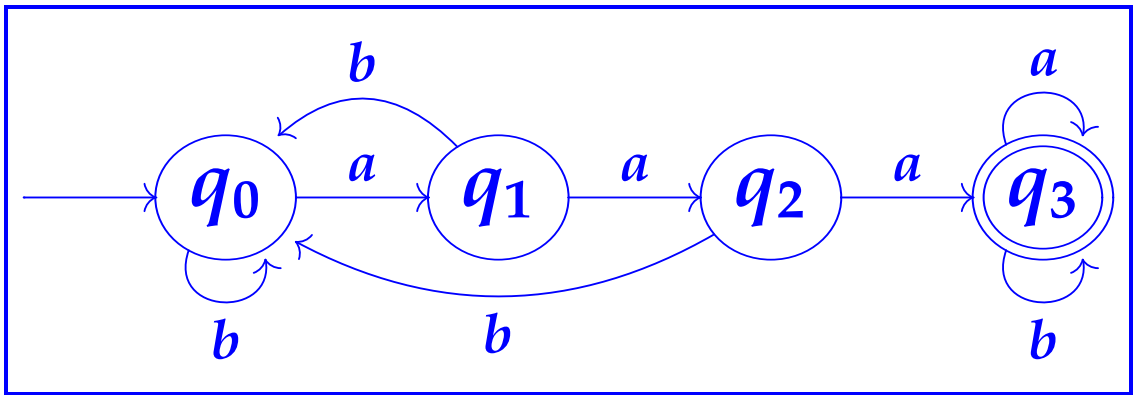
Language accepted by a finite automaton M

- Look at paths in the transition graph from the start state to *some* accepting state.
- Each such path gives a string of input symbols, namely the string of labels on each transition in the path.
- The set of all such strings is by definition **the language accepted by** M, written $L(M)$.

Notation: write $q \xrightarrow{u}{}^{*} q'$ to mean that in the automaton there is a path from state q to state q' whose labels form the string u.

(**N.B.** $q \xrightarrow{\varepsilon}{}^{*} q'$ means $q = q'$.)

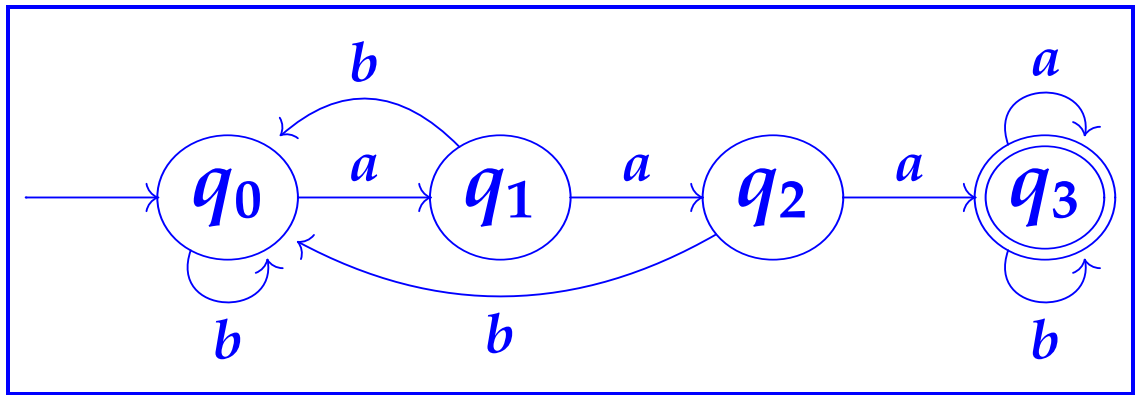
Example of an accepted language

$M \triangleq$ 

For example

- $aaab \in L(M)$, because $q_0 \xrightarrow{aaab}{}^{*} q_3$
- $abaa \notin L(M)$, because $\forall q(q_0 \xrightarrow{abaa}{}^{*} q \Leftrightarrow q = q_2)$

Example of an accepted language

$M \triangleq$ 

Claim:

$$L(M) = L((a|b)^*aaa(a|b)^*)$$

set of all strings matching the regular expression $(a|b)^*aaa(a|b)^*$

(q_i (for $i = 0, 1, 2$) represents the state in the process of reading a string in which the last i symbols read were all as)

Non-deterministic finite automaton (NFA)

is by definition a 5-tuple $M = (Q, \Sigma, \Delta, s, F)$, where:

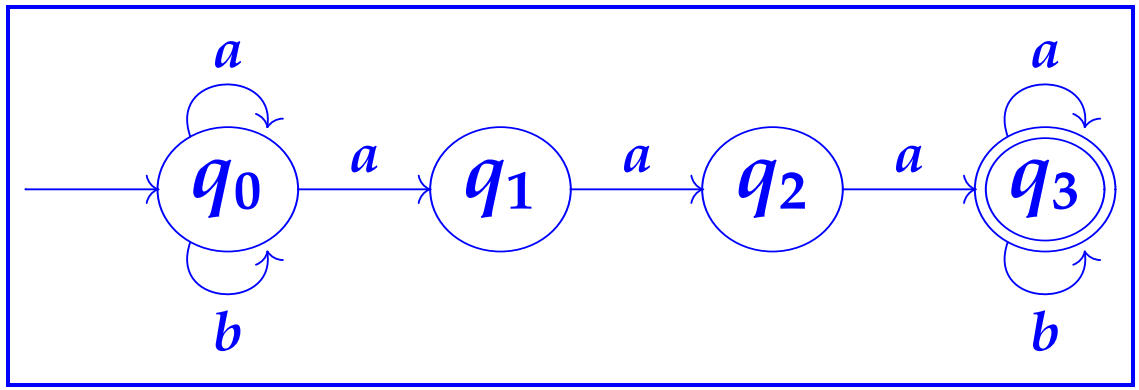
- Q is a finite set (of **states**)
- Σ is a finite set (the alphabet of **input symbols**)
- Δ is a subset of $Q \times \Sigma \times Q$ (the **transition relation**)
- s is an element of Q (the **start state**)
- F is a subset of Q (the **accepting states**)

Notation: write "$q \xrightarrow{a} q'$ in M" to mean $(q, a, q') \in \Delta$.

Example of an NFA

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

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



For example $\{q \mid q_1 \xrightarrow{a} q\} = \{q_2\}$

$\{q \mid q_1 \xrightarrow{b} q\} = \emptyset$

$\{q \mid q_0 \xrightarrow{a} q\} = \{q_0, q_1\}$.

The language accepted by this automaton is the same as for the automaton on Slide 44, namely $\{u \in \{a, b\}^* \mid u$ contains three consecutive a's$\}$.

Deterministic finite automaton (DFA)

A **deterministic finite automaton** (DFA) is an NFA $M = (Q, \Sigma, \Delta, s, F)$ with the property that for each state $q \in Q$ and each input symbol $a \in \Sigma_M$, there is a unique state $q' \in Q$ satisfying $q \xrightarrow{a} q'$.

In a DFA $\Delta \subseteq Q \times \Sigma \times Q$ is the graph of a function $Q \times \Sigma \to Q$, which we write as δ and call the **next-state function**.

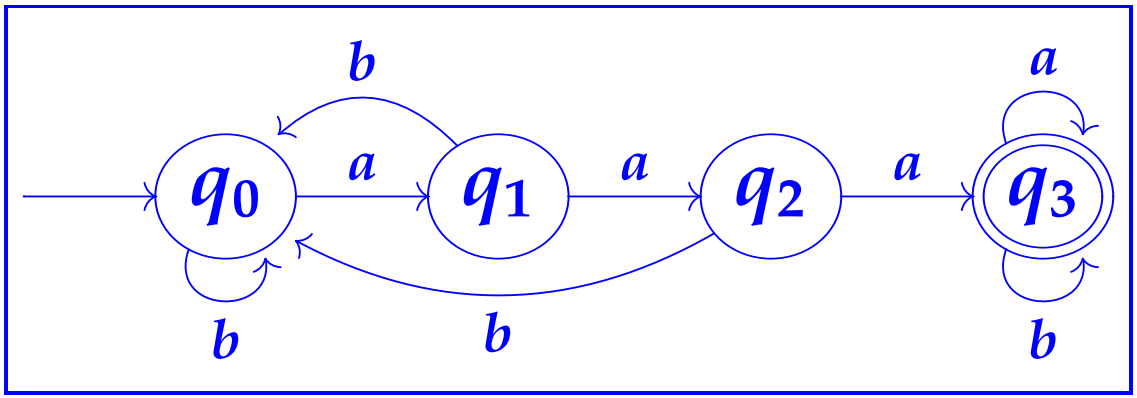
Thus for each (state, input symbol)-pair (q, a), $\delta(q, a)$ is the unique state that can be reached from q by a transition labelled a:

$$\forall q' (q \xrightarrow{a} q' \;\Leftrightarrow\; q' = \delta(q, a))$$

Example of a DFA

with input alphabet $\{a, b\}$

$M \triangleq$

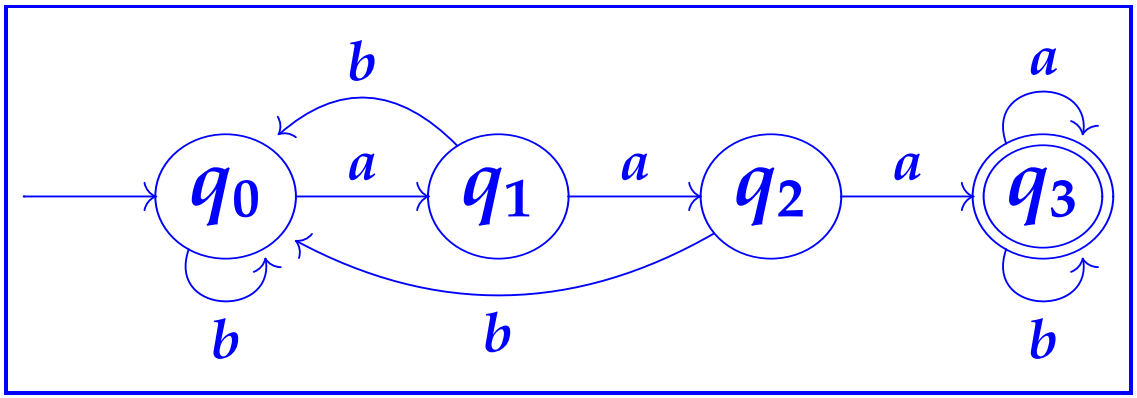


next-state function:

δ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

Example of an NFA

with input alphabet $\{a, b, c\}$

$M \triangleq$ 

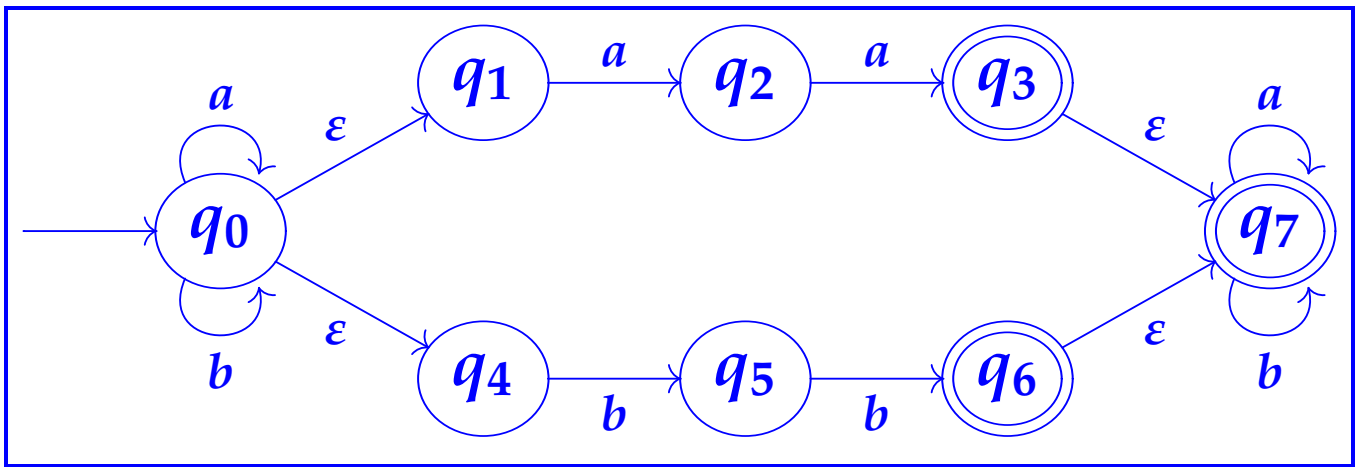
M is non-deterministic, because for example $\{q \mid q_0 \xrightarrow{c} q\} = \emptyset$.

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

$$M = (Q, \Sigma, \Delta, s, F, T)$$

is an NFA $(Q, \Sigma, \Delta, s, F)$ together with a subset $T \subseteq Q \times Q$, called the **ε-transition relation**.

Example:



Notation: write "$q \xrightarrow{\varepsilon} q'$ in M" to mean $(q, q') \in T$.

(**N.B.** for NFA$^\varepsilon$s, we always assume $\varepsilon \notin \Sigma$.)

Language accepted by an NFA$^{\varepsilon}$

$M = (Q, \Sigma, \Delta, s, F, T)$

- Look at paths in the transition graph (including ε-transitions) from start state to *some* accepting state.
- Each such path gives a string in Σ^*, namely the string of non-ε labels that occur along the path.
- The set of all such strings is by definition **the language accepted by** M, written $L(M)$.

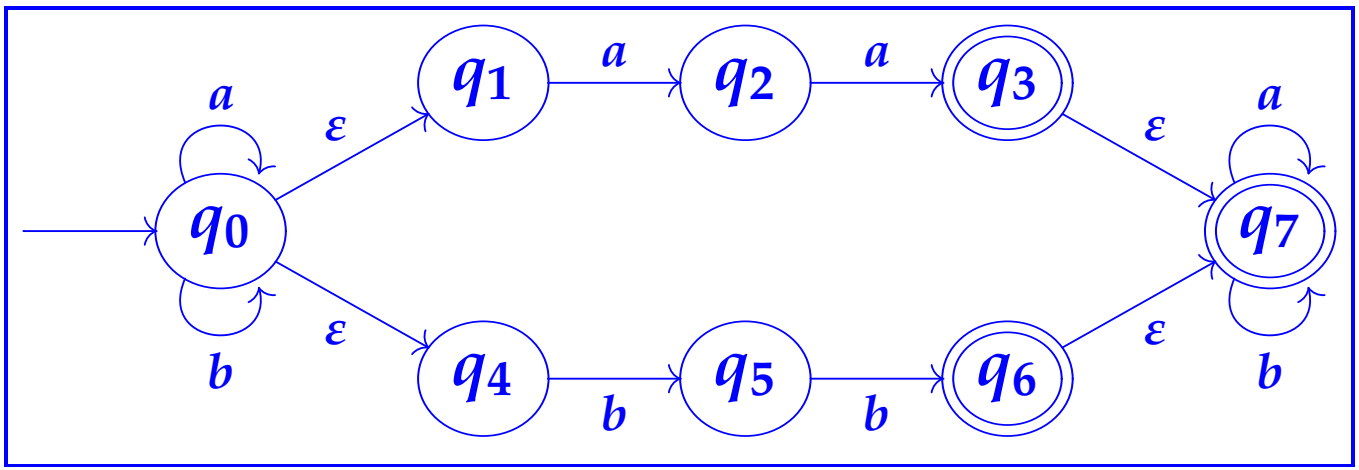
Notation: write $q \stackrel{u}{\Rightarrow} q'$ to mean that there is a path in M from state q to state q' whose non-ε labels form the string $u \in \Sigma^*$.

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

$$M = (Q, \Sigma, \Delta, s, F, T)$$

is an NFA $(Q, \Sigma, \Delta, s, F)$ together with a subset $T \subseteq Q \times Q$, called the **ε-transition relation**.

Example:

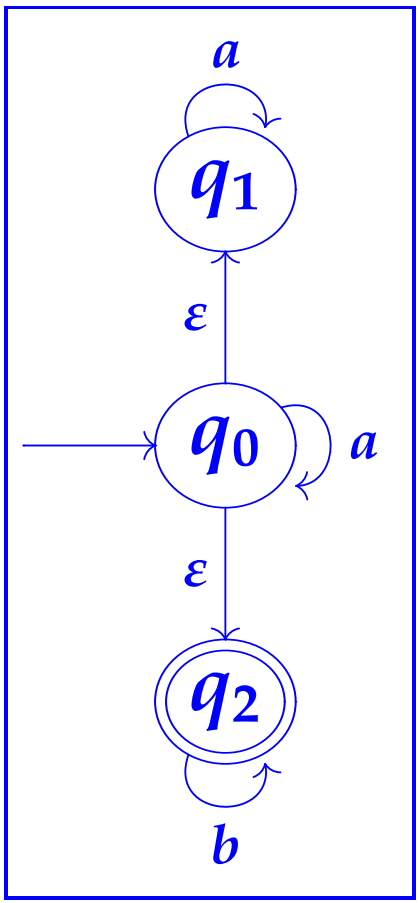


For this NFA$^\varepsilon$ we have, e.g.: $q_0 \stackrel{aa}{\Rightarrow} q_2$, $q_0 \stackrel{aa}{\Rightarrow} q_3$ and $q_0 \stackrel{aa}{\Rightarrow} q_7$.

In fact the language of accepted strings is equal to the set of strings matching the regular expression $(a|b)^*(aa|bb)(a|b)^*$.

Example of the subset construction

M



next-state function for 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$^\varepsilon$ $M = (Q, \Sigma, \Delta, s, F, T)$ there is a DFA $PM = (\mathcal{P}(Q), \Sigma, \delta, s', F')$ accepting exactly the same strings as M, i.e. with $L(PM) = L(M)$.

Definition of PM:

- set of states is the powerset $\mathcal{P}(Q) = \{S \mid S \subseteq Q\}$ of the set Q of states of M
- same input alphabet Σ as for M
- next-state function maps each $(S, a) \in \mathcal{P}(Q) \times \Sigma$ to $\delta(S, a) \triangleq \{q' \in Q \mid \exists q \in S.\ q \stackrel{a}{\Rightarrow} q' \text{ in } M\}$
- start state is $s' \triangleq \{q' \in Q \mid s \stackrel{\varepsilon}{\Rightarrow} q'\}$
- subset of accepting sates is $F' \triangleq \{S \in \mathcal{P}(Q) \mid S \cap F \neq \emptyset\}$

To prove the theorem we show that $L(M) \subseteq L(PM)$ and $L(PM) \subseteq L(M)$.

Regular Languages

Kleene's Theorem

Definition. A language is **regular** iff it is equal to $L(M)$, the set of strings accepted by some deterministic finite automaton M.

Theorem.

(a) For any regular expression r, the set $L(r)$ of strings matching r is a regular language.

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

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$\boxed{L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(iii) **Induction step for $r_1 r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Concat(M_1, M_2)$ satisfying

$$L(Concat(M_1, M_2)) = \{u_1 u_2 \mid u_1 \in L(M_1) \;\&\; u_2 \in L(M_2)\}$$

Thus $L(r_1 r_2) = L(Concat(M_1, M_2))$ when $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$.

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(iii) **Induction step for $r_1 r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Concat(M_1, M_2)$ satisfying

$$L(Concat(M_1, M_2)) = \{u_1 u_2 \mid u_1 \in L(M_1) \;\&\; u_2 \in L(M_2)\}$$

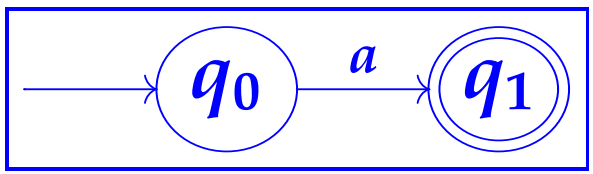
Thus $L(r_1 r_2) = L(Concat(M_1, M_2))$ when $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$.

(iv) **Induction step for r^*:** given NFA$^\varepsilon$ M, construct an NFA$^\varepsilon$ $Star(M)$ satisfying

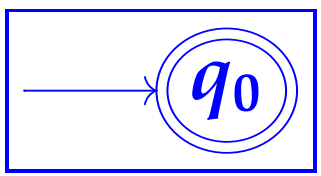
$$L(Star(M)) = \{u_1 u_2 \ldots u_n \mid n \geq 0 \text{ and each } u_i \in L(M)\}$$

Thus $L(r^*) = L(Star(M))$ when $L(r) = L(M)$.

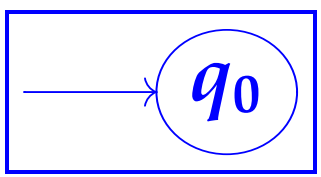
NFAs for regular expressions a, ϵ, $\emptyset$



just accepts the one-symbol string a

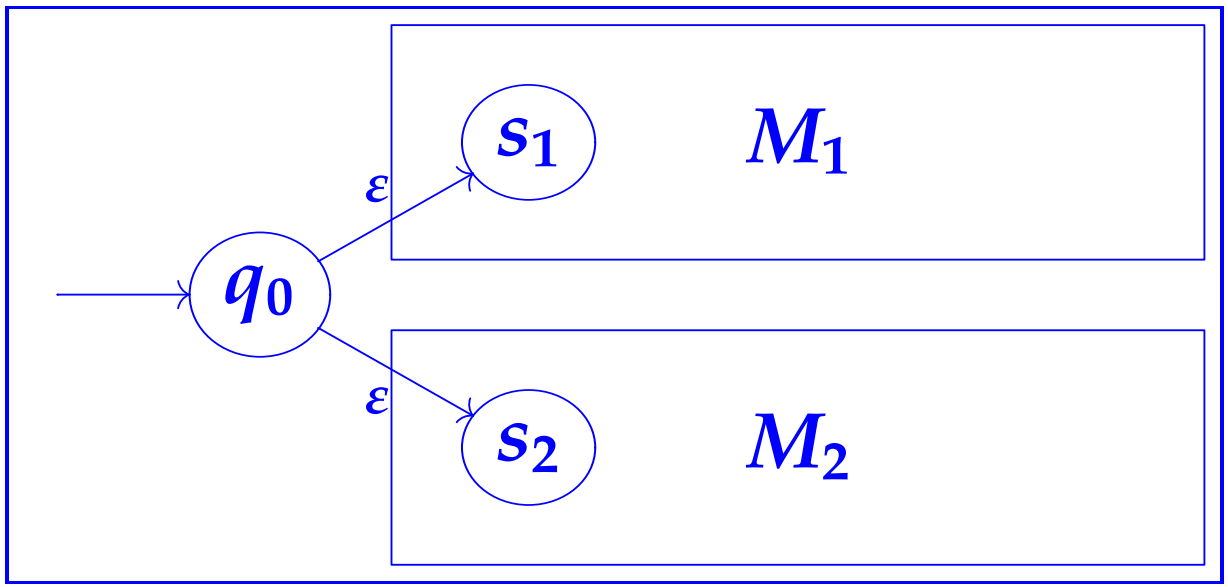


just accepts the null string, ε



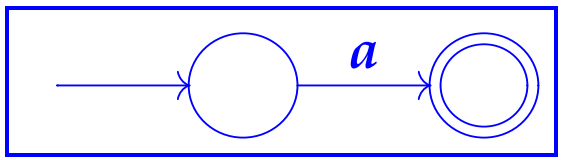
accepts no strings

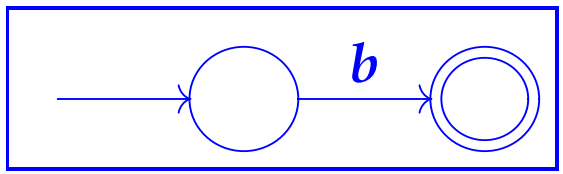
$Union(M_1, M_2)$

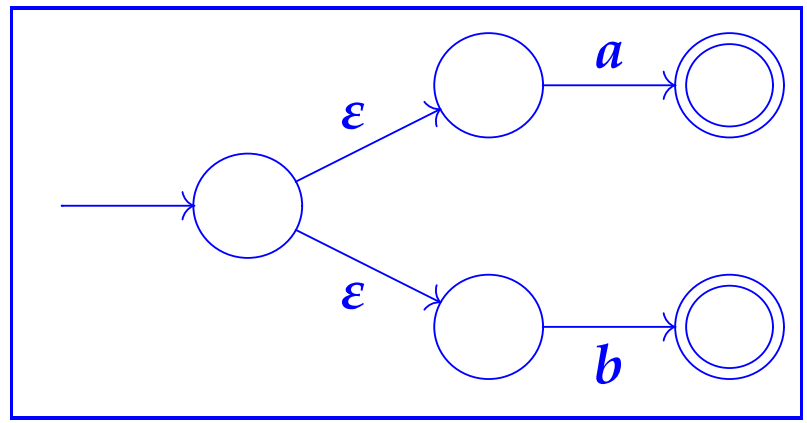


accepting states = union of accepting states of M_1 and M_2

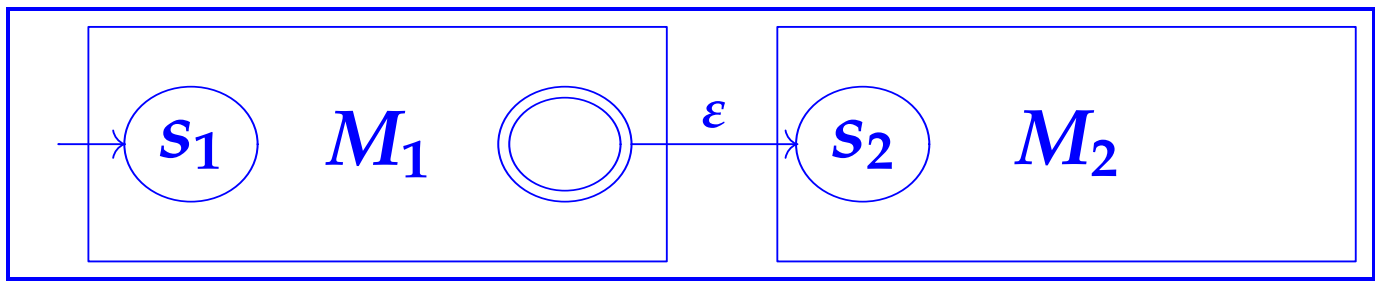
For example,

if $M_a =$ 

and $M_b =$ 

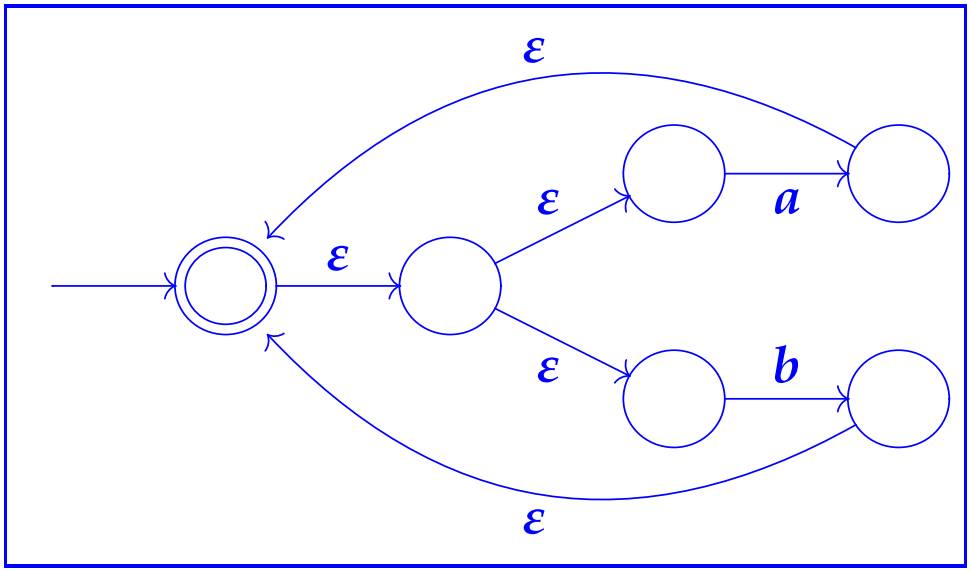
then $Union(M_a, M_b) =$ 

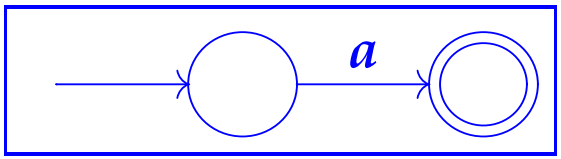
$Concat(M_1, M_2)$

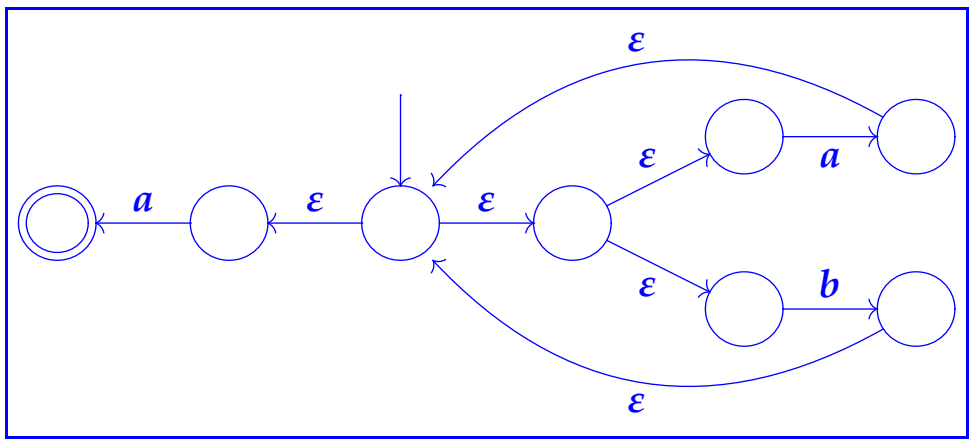


accepting states are those of M_2

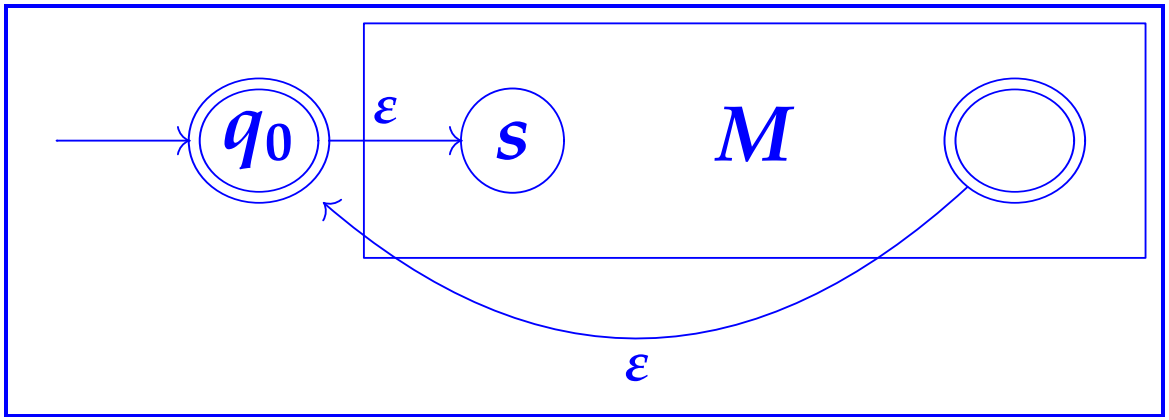
For example,

if $M_1 =$ 

and $M_2 =$ 

then $Concat(M_1, M_2) =$ 

$Star(M)$

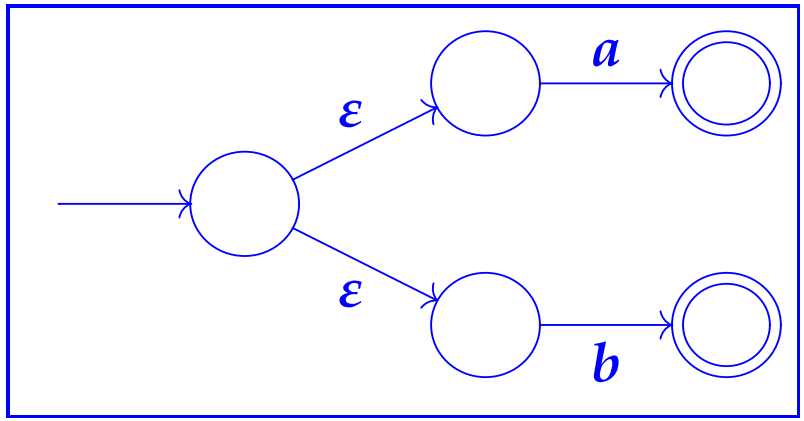


the only accepting state of $Star(M)$ is q_0

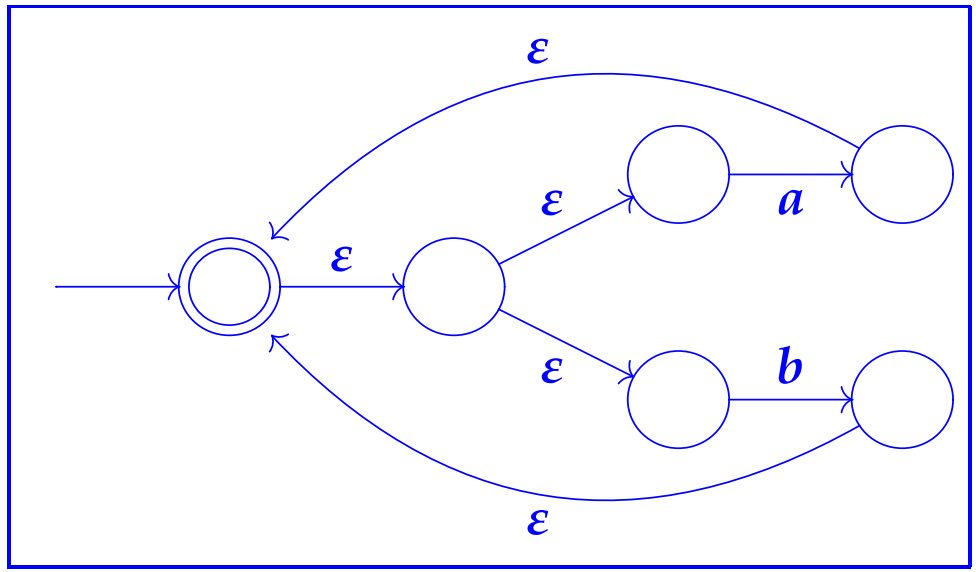
(N.B. doing without q_0 by just looping back to s
and making that accepting won't work – Exercise 4.1.)

For example,

if $M =$



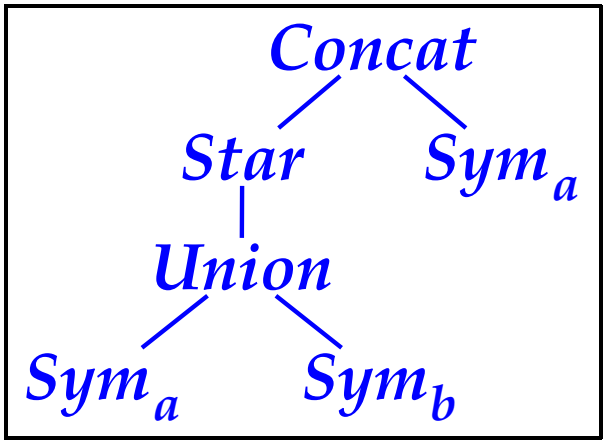
then $Star(M) =$



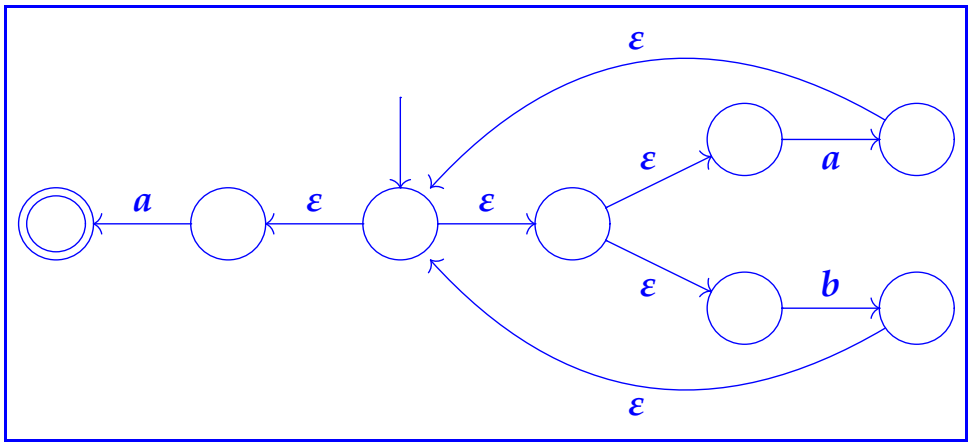
Example

Regular expression $(a|b)^*a$

whose abstract syntax tree is



is mapped to the NFA$^\varepsilon$ $Concat(Star(Union(M_a, M_b)), M_a) =$



(*cf.* Slides 68, 71 and 74).

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, 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, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

Decidability of matching

We now have a positive answer to question (a) on Slide 38. Given string u and regular expression r:

- construct an NFA$^\varepsilon$ M satisfying $L(M) = L(r)$;
- in PM (the DFA obtained by the subset construction, Slide 59) carry out the sequence of transitions corresponding to u from the start state to some state q (because PM is deterministic, there is a unique such transition sequence);
- check whether q is accepting or not: if it is, then $u \in L(PM) = L(M) = L(r)$, so u matches r; otherwise $u \notin L(PM) = L(M) = L(r)$, so u does not match r.

(The subset construction produces an exponential blow-up of the number of states: PM has 2^n states if M has n. This makes the method described above potentially inefficient – more efficient algorithms exist that don't construct the whole of PM.)

Kleene's Theorem

Definition. A language is **regular** iff it is equal to $L(M)$, the set of strings accepted by some deterministic finite automaton M.

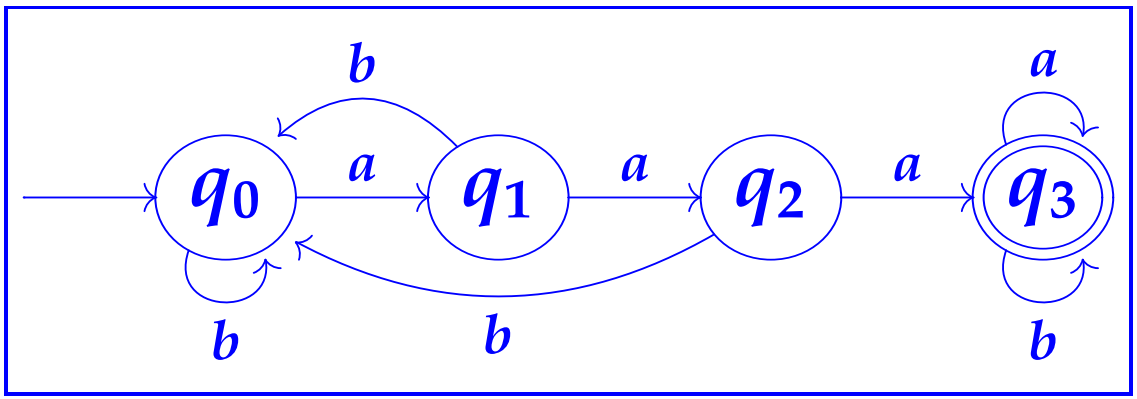
Theorem.

(a) For any regular expression r, the set $L(r)$ of strings matching r is a regular language.

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

Example of a regular language

Recall the example DFA we used earlier:

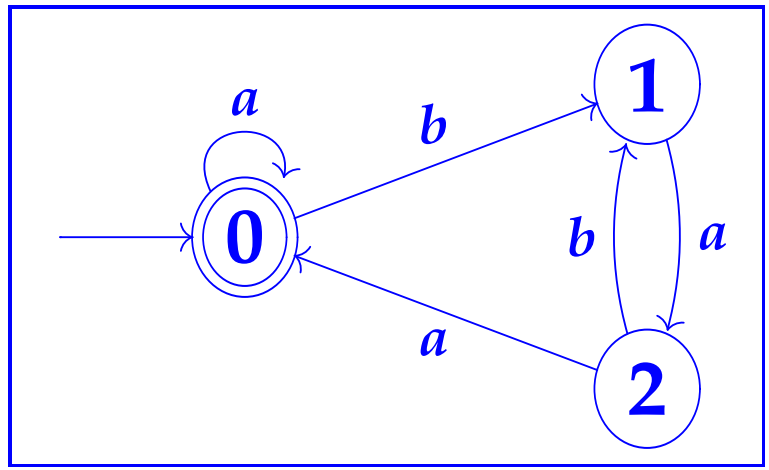
$M \triangleq$ 

In this case it's not hard to see that $L(M) = L(r)$ for

$$r = (a|b)^* aaa (a|b)^*$$

Example

$M \triangleq$

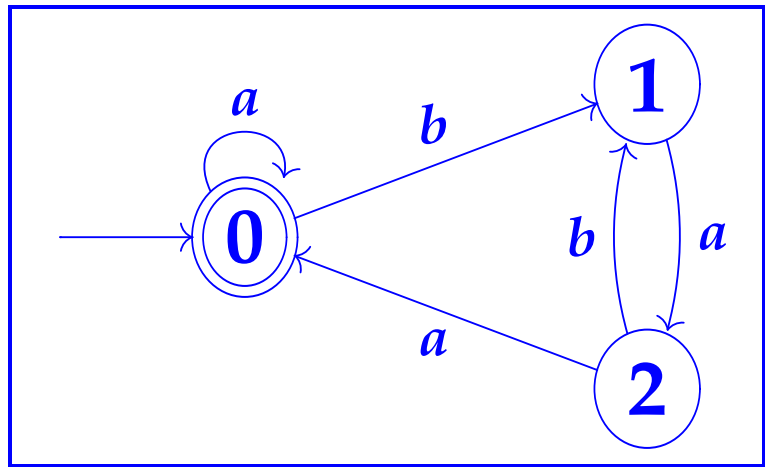


$L(M) = L(r)$ for which regular expression r?

Guess: $r = a^*|a^*b(ab)^*aaa^*$

Example

$M \triangleq$



$L(M) = L(r)$ for which regular expression r?

Guess: $r = a^*|a^*b(ab)^*aaa^*$

WRONG! since $baabaa \in L(M)$
but $baabaa \notin L(a^*|a^*b(ab)^*aaa^*)$

We need an algorithm for constructing a suitable r for each M (plus a proof that it is correct).

Lemma. Given an NFA $M = (Q, \Sigma, \Delta, s, F)$, for each subset $S \subseteq Q$ and each pair of states $q, q' \in Q$, there is a regular expression $r^S_{q,q'}$ satisfying

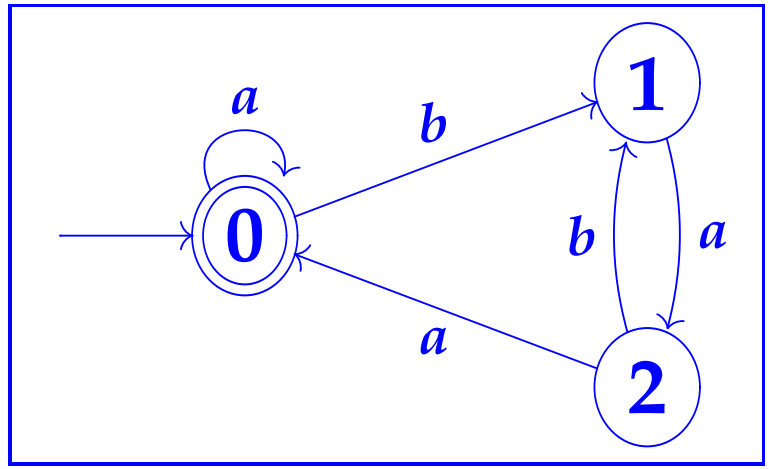
$$L(r^S_{q,q'}) = \{u \in \Sigma^* \mid q \xrightarrow{u}{}^* q' \text{ in } M \text{ with all intermediate states of the sequence of transitions in } S\}.$$

Hence if the subset F of accepting states has k distinct elements, $q_1, \ldots, q_k$ say, then $L(M) = L(r)$ with $r \triangleq r_1 | \cdots | r_k$ where

$$r_i = r^Q_{s,q_i} \qquad (i = 1, \ldots, k)$$

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

$M \triangleq$



By direct inspection we have:

$r_{i,j}^{\{0\}}$	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			

(we don't need the unfilled entries in the tables)

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, 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, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

$Not(M)$

Given DFA $M = (Q, \Sigma, \delta, s, F)$,
then $Not(M)$ is the DFA with

- set of states $= Q$
- input alphabet $= \Sigma$
- next-state function $= \delta$
- start state $= s$
- accepting states $= \{q \in Q \mid q \notin F\}$.

(i.e. we just reverse the role of accepting/non-accepting and leave everything else the same)

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

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

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs M_1 and M_2, then $L_1 \cap L_2 = L(Not(PM))$ where M is the NFA$^\varepsilon$ $Union(Not(M_1), Not(M_2))$. □

[It is not hard to directly construct a DFA $And(M_1, M_2)$ from M_1 and M_2 such that $L(And(M_1, M_2)) = L(M_1) \cap L(M_2)$ – see Exercise 4.7.]

Regular languages are closed under intersection

Corollary: given regular expressions r_1 and r_2,there is a regular expression, which we write as $r_1 \,\&\, r_2$, such that

> a string u matches $r_1 \,\&\, r_2$ iff it matches both r_1 and r_2.

Proof. By Kleene (a), $L(r_1)$ and $L(r_2)$ are regular languages and hence by the theorem, so is $L(r_1) \cap L(r_2)$. Then we can use Kleene (b) to construct a regular expression $r_1 \,\&\, r_2$ with $L(r_1 \,\&\, r_2) = L(r_1) \cap L(r_2)$. □

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, 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, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

Equivalent regular expressions

Definition. Two regular expressions r and s are said to be **equivalent** if $L(r) = L(s)$, that is, they determine exactly the same sets of strings via matching.

For example, are $b^*a(b^*a)^*$ and $(a|b)^*a$ equivalent?

Equivalent regular expressions

Definition. Two regular expressions r and s are said to be **equivalent** if $L(r) = L(s)$, that is, they determine exactly the same sets of strings via matching.

For example, are $b^*a(b^*a)^*$ and $(a|b)^*a$ equivalent?

Answer: yes (Exercise 2.3)

How can we decide all such questions?

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$
iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$
iff $L((\sim r) \,\&\, s) = \emptyset = L((\sim s) \,\&\, r)$
iff $L(M) = \emptyset = L(N)$

where M and N are DFAs accepting the sets of strings matched by the regular expressions $(\sim r) \,\&\, s$ and $(\sim s) \,\&\, r$ respectively.

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$
iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$
iff $L((\sim r)\,\&\,s) = \emptyset = L((\sim s)\,\&\,r)$
iff $L(M) = \emptyset = L(N)$

where M and N are DFAs accepting the sets of strings matched by the regular expressions $(\sim r)\,\&\,s$ and $(\sim s)\,\&\,r$ respectively.

So to decide equivalence for regular expressions it suffices to

check, given any given DFA M, whether or not it accepts some string.

Note that the number of transitions needed to reach an accepting state in a finite automaton is bounded by the number of states (we can remove loops from longer paths). So we only have to check finitely many strings to see whether or not $L(M)$ is empty.

The Pumping Lemma

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, 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, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

Examples of languages that are not regular

- 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 $|w| \geq \ell$ can be expressed as a concatenation of three strings, $w = u_1 v u_2$, where u_1, v and u_2 satisfy:

- $|v| \geq 1$
  (i.e. $v \neq \varepsilon$)
- $|u_1 v| \leq \ell$
- 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.)

Suppose $L = L(M)$ for a DFA $M = (Q, \Sigma, \delta, s, F)$. Taking ℓ to be the number of elements in Q, if $n \geq \ell$, then in

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

$q_0, \ldots, q_\ell$ 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

$$s = q_0 \xrightarrow{u_1\,*} q_i \overset{v}{=} q_j \xrightarrow{u_2\,*} q_n \in F$$

where

$$u_1 \triangleq a_1 \ldots a_i \qquad v \triangleq a_{i+1} \ldots a_j \qquad u_2 \triangleq a_{j+1} \ldots a_n$$

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

no matter how w is split into three, $w = u_1 v u_2$,
with $|u_1 v| \leq \ell$ and $|v| \geq 1$, there is some $n \geq 0$
for which $u_1 v^n u_2$ is *not* in L

} (†)

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$

[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†) on Slide 104.]

(ii) $L_2 \triangleq \{w \in \{a,b\}^* \mid w \text{ a palindrome}\}$

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

(iii) $L_3 \triangleq \{a^p \mid p \text{ prime}\}$

[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 with the pumping lemma property

$$L \triangleq \{c^m a^n b^n \mid m \geq 1 \,\&\, n \geq 0\} \cup \{a^m b^n \mid m, n \geq 0\}$$

satisfies the pumping lemma property on Slide 101 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 5.1.