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

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

abstract syntax trees

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

rules:

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

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

(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

grep global regular expression search and print tool

Unix tool which searches a file for matches to a regular expression. Can print various things:

Print lines which contain matches:

```
grep <reg exp> <file name>
```

Print the (maximal) matching strings:

```
grep -o <reg exp> <file name>
```

Print number of lines which contain matches:

```
grep  -c <reg exp> <file name>
```

unix command line:

```
> more foo
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
>
```

foo:

```
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
```

unix command line:

```
> grep 'apple' foo
apple
pineapple
>
```

foo:

```
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
```

unix command line:

```
> grep 'apple\|berry' foo
apple
pineapple
blueberry
loganberry
cranberry
>
```

foo:

```
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
```

unix command line:

```
> grep -o 'pp*' foo
pp
p
pp
> grep 'pp*' foo
apple
pineapple
>
```

foo:

```
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
```

unix command line:

```
> grep -o 'ap*le' foo
apple
apple
ale
>
```

foo:

```
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
```

unix command line:

```
> grep '[a-c][o-r]' foo
apple
pineapple
cranberry
> grep -o '[a-c][o-r]' foo
ap
ap
cr
>
```

foo:

```
A list of fruit is not terribly exciting:
apple
pineapple
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
```

unix command line:

```
> grep -v 'apple' foo
A list of fruit is not terribly exciting:
blueberry
loganberry
cranberry
orange (which nothing rhymes with)
valencia orange
tangerine
etc
>
```

Regular expressions (concrete syntax)

over a given alphabet Σ.

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

In theory, practice and theory are the same thing.

In practice they rarely are.

Regular expressions (concrete syntax)

over a given alphabet Σ.

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

In theory, practice and theory are the same thing.

In practice they rarely are.

e.g. <u>grep</u> has to deal with the fact that Σ' can't be disjoint from Σ

Questions Computer Scientists ask

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

in other words, decides, for any r, whether $u \in L(r)$

Questions Computer Scientists ask

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

in other words, decides, for any r, whether $u \in L(r)$

An algorithm? what's an algorithm? I mean what is it in a mathematical sense?

Questions Computer Scientists ask

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

in other words, decides, for any r, whether $u \in L(r)$

An algorithm? what's an algorithm? I mean what is it in a mathematical sense?

leads us to define automata which "execute algorithms"

Questions Computer Scientists ask

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

in other words, decides, for any r, whether $u \in L(r)$

An algorithm? what's an algorithm? I mean what is it in a mathematical sense?

leads us to define automata which "execute algorithms"
next chunk of the course...

Questions Computer Scientists ask

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

Questions Computer Scientists ask

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

Yes

Questions Computer Scientists ask

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

Yes

Yes because there are convenient notations like $[a-z]$ to mean $a|b|c\ldots|z$ and complement, $\sim r$, which is defined to match all strings that r does not.

Questions Computer Scientists ask

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

Yes and No

Yes because there are convenient notations like $[a-z]$ to mean $a|b|c\dots|z$ and complement, $\sim r$, which is defined to match all strings that r does not.

Questions Computer Scientists ask

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

Yes and No

Yes because there are convenient notations like $[a - z]$ to mean $a|b|c\dots|z$ and complement, $\sim r$, which is defined to match all strings that r does not.

No because such conveniences don't allow us to define languages we can't already define

Questions Computer Scientists ask

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

Yes and No

Yes because there are convenient notations like $[a - z]$ to mean $a|b|c\ldots|z$ and complement, $\sim r$, which is defined to match all strings that r does not.

No because such conveniences don't allow us to define languages we can't already define

Why not include them in our basic definition??

Questions Computer Scientists ask

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

Yes and No

Yes because there are convenient notations like $[a - z]$ to mean $a|b|c \ldots |z$ and complement, $\sim r$, which is defined to match all strings that r does not.

No because such conveniences don't allow us to define languages we can't already define

Why not include them in our basic definition??

Because they give us more rules to analyse!

Questions Computer Scientists ask

(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?

We will answer this when we answer (a).

Questions Computer Scientists ask

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

Pretty clearly no.

Questions Computer Scientists ask

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

Pretty clearly no.

in fact even simple languages like $a^n b^n, \forall n \in \mathbb{N}$ or well-bracketed arithmetic expressions are not regular

Questions Computer Scientists ask

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

Pretty clearly no.

in fact even simple languages like $a^n b^n, \forall n \in \mathbb{N}$ or well-bracketed arithmetic expressions are not regular

we will derive and use the Pumping Lemma to show this

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

We are about to describe some different types of finite automata.

The game plan is as follows:

- define (non-deterministic) finite automata in general

We are about to describe some different types of finite automata.

The game plan is as follows:

- define (non-deterministic) finite automata in general
- define deterministic finite automata (as a special case)

We are about to describe some different types of finite automata.

The game plan is as follows:

- define (non-deterministic) finite automata in general
- define deterministic finite automata (as a special case)
- define non-deterministic finite automata with ε-transitions

We are about to describe some different types of finite automata.

The game plan is as follows:

- define (non-deterministic) finite automata in general
- define deterministic finite automata (as a special case)
- define non-deterministic finite automata with ε-transitions
- show that from any non-deterministic finite automaton with ε-transitions we can mechanically produce an equivalent deterministic finite automaton

Why?

- we are claiming that a deterministic finite automata (DFA) is an embodiment of an algorithm

Why?

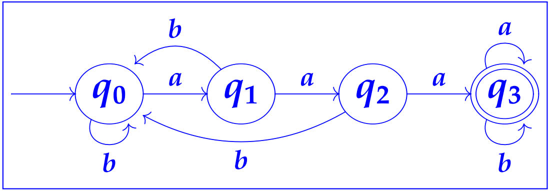
- we are claiming that a deterministic finite automata (DFA) is an embodiment of an algorithm
- non-deterministic finite automata with ε-transitions (NFA$^{\varepsilon}$'s) map on to our problem (matching regular expressions) more naturally ...

Why?

- we are claiming that a deterministic finite automata (DFA) is an embodiment of an algorithm
- non-deterministic finite automata with ε-transitions (NFA$^\varepsilon$'s) map on to our problem (matching regular expressions) more naturally ...
- ...so we will produced the NFA$^\varepsilon$'s we want and then rely on the fact that for each there is an equivalent DFA.

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

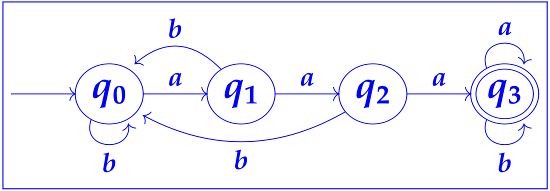
Language accepted by a finite automaton M

- 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 $q = q'$.)

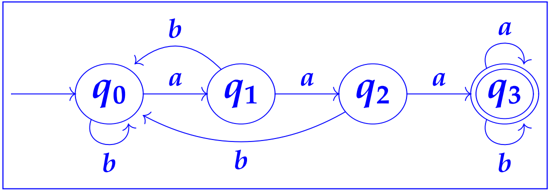
Example of an accepted language

$M \triangleq$ 

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 a's)

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 $(q, a, q') \in \Delta$.

Why do we say this is non-deterministic?

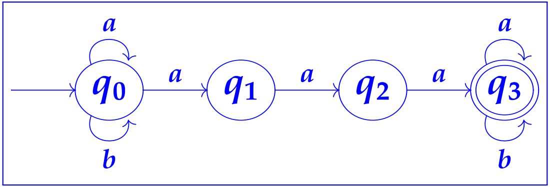
Δ, the transition relation specifies a set of next states for a given current state and given input symbol.

That set might have 0, 1 or more elements.

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

So we define a deterministic finite automata so that Δ is restricted to specify exactly <u>one</u> next state for any given state and input symbol

we do this by saying the relation Δ has to be a function δ from $Q \times \Sigma$ to Q

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**.

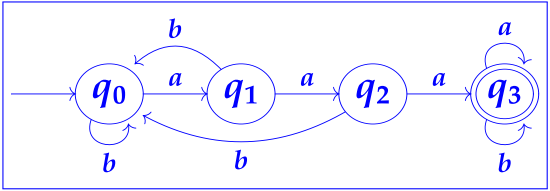
Thus for each (state, input symbol)-pair (q, a), $\delta(q, a)$ is the unique state that can be reached from q by a transition labelled a:

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

Example of a DFA

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

$M \triangleq$



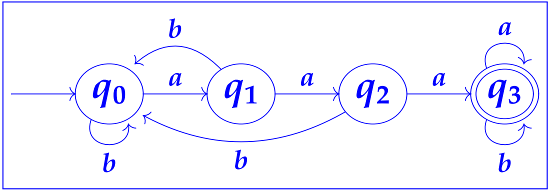
next-state function:

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 $\{q \mid q_0 \xrightarrow{c} q\} = \emptyset$.

so alphabet matters!

Now let's make things a bit more interesting (well complicated) . . .

We are going to introduce a new form of transition, an ε-transition which allows us to more from one state to another without reading a symbol.

These (in general) introduce non-determinism all by themselves.

ε-Transitions

When constructing machines for matching strings with regular expressions (as we will do later), it is useful to consider finite state machines exhibiting an 'internal' form of non-determinism in which the machine is allowed to change state without consuming any input symbol. One calls such transitions **ε-transitions** and writes them as $q \xrightarrow{\varepsilon} q'$. This leads to the definition on Slide 86.

When using an NFA$^\varepsilon$ M to accept a string $u \in \Sigma^*$ of input symbols, we are interested in sequences of transitions in which the symbols in u occur in the correct order, but with zero or more ε-transitions before or after each one. We write $q \stackrel{u}{\Rightarrow} q'$ to indicate that such a sequence exists from state q to state q' in the NFA$^\varepsilon$. Equivalently, $\{(q,u,q') \mid q \stackrel{u}{\Rightarrow} q'\}$ is the subset of $Q \times \Sigma^* \times Q$ inductively defined by

$$\boxed{\text{axioms: } \frac{}{(q,\varepsilon,q)} \quad \text{and rules: } \frac{(q,u,q')}{(q,u,q'')} \text{if } q' \xrightarrow{\varepsilon} q'', \quad \frac{(q,u,q')}{(q,ua,q'')} \text{if } q' \xrightarrow{a} q''} \text{ (see Exercise 7)}$$

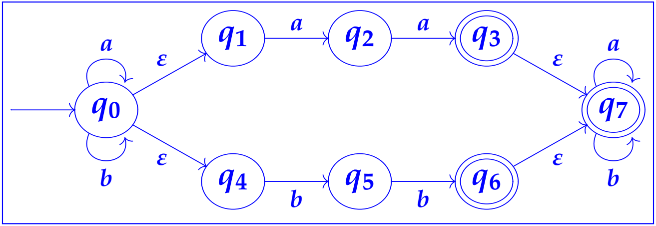
Slide 87 uses the relation $q \stackrel{u}{\Rightarrow} q'$ to define the language accepted by an NFA$^\varepsilon$. For example, for the NFA$^\varepsilon$ on Slide 86 it is not too hard to see that the language accepted consists of all strings which either contain two consecutive a's or contain two consecutive b's, i.e. the language determined by the regular expression $(a|b)^*(aa|bb)(a|b)^*$.

An **NFA with ε-transitions** (NFA^{ε})

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

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^{ε}s, we always assume $\varepsilon \notin \Sigma$.)