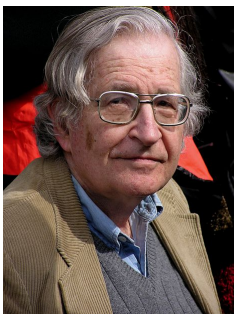
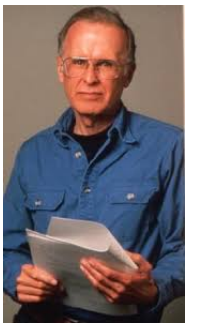
Regular and context-free languages



Noam Chomsky

(linguist)



John Backus

(programmer)

Finite representation of languages

Consider the following language: * = all words in this alphabet

L = all strings from $\{0,1\}^*$ that have two or three occurrences of 1, the first and second of which are not consecutive

Is there a more **'concise'** way of representing this language?

- strings in L can start with any number (possibly none) of 0's: $\{0\}^*$
- then comes the first 1, which must be followed by at least one 0: $\{0\}^*10$
- then any number (possibly none) of 0's: $\{0\}^*10\{0\}^*$
- then comes the second 1: $\{0\}^*10\{0\}^*1$
- then either there are no more 1's but there can be any number (possibly none) of 0's, or there is a third 1 followed by any number (possibly none) of 0's:

$\{0\}^*10\{0\}^*1(\{0\}^*$ or $\{0\}^*1\{0\}^*)$

We will dispense with the braces $\{\}$ and write $\cup$ instead of 'or':

$0^*100^*1(0^* \cup 0^*10^*)$ **regular expression** describing L

Regular expressions: what for?

- They are used in any task that requires *pattern matching*:
  - search engines,
  - protein analysis in bioinformatics,
  - search and replace functionalities in word processors and text editors,
  - text processing utilities like `sed` and `awk` in Unix and Linux.
- Many programming languages provide various regular expression capabilities.
- Lexical analysers in compilers also use regular expressions.
- ...

From language to regular expression: Example 2

L = all the strings consisting of some number (possibly none) of a's, followed by some number (possibly none) of b's

$= \{\varepsilon, a, b, aa, bb, ab, aaa, aab, bbb, abb, \ldots, aaaabbbbbbb, \ldots\}$

Let's try: $\underline{\{a\}^*\{b\}^*}$

(a U b)* is equal to {a,b}*

$\rightsquigarrow$ L is represented by the regular expression a^*b^*

this is equal to {a,b}*ba{a,b}*

Notation: $\boxed{L = L[a^*b^*]}$

Describe the words in the alphabet $\{a, b\}$ that are **not** in $L[a^*b^*]$

all words containing a sub-word ba: $(a \cup b)^*ba(a \cup b)^*$

From regular expression to language

Example 3:

$L[a(a^* \cup b^*)]$ = all words starting with a, followed by either a
(possibly empty) word of a's, or
a (possibly empty) word of b's

$= \{a, aa, ab, aaa, abb, aaaa, abbb, aaaaaa, abbbbbbb, \dots\}$

Example 4:

$L[a(a \cup b)^*]$ = all words starting with a, followed by any word over $\{a, b\}$

= all words over the alphabet $\{a, b\}$ starting with a

$= \{a, aa, ab, aaa, aab, aba, abb, aaaaaa, abbaabab, aabbababba, \dots\}$

These two examples are NOT the same !

From regular expression to language: Example 5

$$L[(b \cup aaa^*)^*] \;=\; ?$$

- Let's start with $L[aaa^*]$: all words of a's of length ≥ 2

  $$= \{aa, aaa, aaaa, aaaaa, \dots\}$$

- $L[b \cup aaa^*]$: as above, plus the word b

  $$= \{b, aa, aaa, aaaa, aaaaa, \dots\}$$

- $L[(b \cup aaa^*)^*]$: we can take any number (possibly none) of words from $L[b \cup aaa^*]$ and concatenate them

– Is there a word over $\{a, b\}$ that is not in $L[(b \cup aaa^*)^*]$?

– Do the words ε, $aabbaaabab$, and $bbbabbaaabaa$ belong to the language $L[(b \cup aaa^*)^*]$?

Regular expressions: formal definition

Every regular expression (over an alphabet Σ) is a string consisting of symbols from Σ, plus symbols from the list $\cup$, $*$, $($, $)$, ε, $\emptyset$

Inductive definition of the set of **regular expressions** (over the alphabet Σ):

ε is empty words

- $\emptyset$, ε and each symbol in Σ are regular expressions
- if α and β are regular expressions, then so are $(\alpha \cup \beta)$, $(\alpha\beta)$, and (α^*)
- no other string is a regular expression

Examples: $\emptyset$, ε, $((a^*)(b^*))$, $(((a((a \cup b)^*))b)(b^*))$,

$(((x(y^*))y)(y \cup z))$, $((\square \cup (\diamond^*))((\diamond\square)^*))$

Is $\cup^* a^*$ a regular expression?

Regular expressions: notational conventions

These brackets are pretty incomprehensible.

So we introduce some conventions:

- We omit the outermost brackets
- We omit the brackets when concatenate expressions

  E.g., we write $abbabb$ instead of $(((ab)(ab))b)$
- We agree that * 'binds tighter' than concatenation and $\cup$

  E.g., we write aab^* instead of $(aa)(b^*)$
- We agree that concatenation binds tighter than $\cup$

Examples: (cf. the previous slide)

$$a^*b^*, \quad a(a \cup b)^*bb^*, \quad xy^*y(y \cup z), \quad (\square \cup \diamond^*)(\diamond\square)^*$$

But take care: say, $(aa)^*b$ and aa^*b are NOT the same!

Always use brackets if in any doubt!

Example 6

$L[(b \cup a)aa^*] = ?$

- $L[b \cup a] = \{a, b\}$
- $L[(b \cup a)a] = \{aa, ba\}$
- $L[(b \cup a)aa^*] =$ all words starting with aa followed by a (possibly empty) word of a's, and all words starting with ba followed by a (possibly empty) word of a's

Compare this language with $L[b \cup aaa^*]$ (see Example 5)

Brackets DO MATTER: the two languages are NOT the same!

Regular expressions specify languages

Every regular expression over Σ **represents** a language over Σ

These are two different things:

a regular expression α	$\leftrightarrow$	the language α represents: $L[\alpha]$
a string consisting of symbols from Σ and $\cup, *, (,), \varepsilon, \emptyset$		a set of words over Σ

For example,

$a(a \cup b)^*$

$L[a(a \cup b)^*] =$ all words over $\{a, b\}$ that start with a

How regular expressions specify languages

Inductive definition

of the **language $L[\alpha]$ represented by the regular expression α:**

- $L[\emptyset] = \emptyset$, $L[\varepsilon] = \{\varepsilon\}$, and $L[a] = \{a\}$ for any symbol a in Σ.
- If α and β are regular expressions, then
  - $L[\alpha \cup \beta]$ = all words that belong either to $L[\alpha]$ or to $L[\beta]$,
  - $L[\alpha\beta]$ = any word from $L[\alpha]$ followed by any word from $L[\beta]$,
  - $L[\alpha^*]$ = any word from $L[\alpha]$ followed by any word from $L[\alpha]$ followed by any word from $L[\alpha]$ ... the number of iterations is arbitrary (possibly none)

Example 7: identifiers in programming languages

Any such identifier begins with a letter, which may be followed by a string consisting of letters and numeric digits.

A regular expression representing the set of all identifiers is:

$$\Big([a-z] \cup [A-Z]\Big)\Big([a-z] \cup [A-Z] \cup [0-9]\Big)^*$$

where

$[a-z]$ is a shorthand for $(a \cup b \cup c \cup \cdots \cup z)$

$[A-Z]$ is a shorthand for $(A \cup B \cup C \cup \cdots \cup Z)$

$[0-9]$ is a shorthand for $(0 \cup 1 \cup 2 \cup \cdots \cup 9)$

Lexical analyser: a part of every compiler which finds all identifiers in the text of the source program. It uses a list of regular expressions to do so.

Example 8: Search engines on the WWW

Notice the similarity between regular expressions and the form of the queries used by search engines on the WWW. Search queries often include `wild cards.'

Some search engines support two wildcards. The asterisk (*) is used to replace multiple characters and the percent (%) symbol is used to replace only one character.

These are easy to simulate by regular expressions: For example, an arbitrary lowercase letter is captured by the regular expression $a \cup b \cup \cdots \cup z$. So to find, say, all Web pages that contain an occurrence of `*turtle*' following (not necessarily immediately) an instance of `*purple,*' one constructs a finite automaton to recognise the language

$$\textit{purple}\ \ (a \cup b \cup \cdots \cup z)^*\ \ \textit{turtle}$$

Every page matching the query will be accepted by this automaton, and any page that doesn't match the query will not be accepted.

Example 9

$L[\varepsilon \cup c^*(a \cup bc^*)] \;=\; ?$

all the strings that are either empty or start with some (possibly none) c's, followed by either an a, or a b followed by some (possibly none) c's

$= \{\varepsilon, a, b, bc, bcc, ca, cb, cbcc, \ldots\}$

NOT in $L[\varepsilon \cup c^*(a \cup bc^*)]$: $ab,\ aa,\ caac,\ \ldots$

Regular languages

A **regular language** is any language that is described by a regular expression.

In other words, a language L is **regular** if $L = L[\alpha]$,
for some regular expression α

For instance, all the languages in Examples 1–9 above are regular.

NOT every language is regular !

Regular languages and finite automata

(1) **There is a general 'mechanical' procedure $\mathcal{A}$ that converts any regular language $L[\alpha]$ to an NFA A such that $L(A) = L[\alpha]$.**

Moreover, there is a general way back:

(2) **There is a general 'mechanical' procedure $\mathcal{B}$ for converting an automaton A into a regular expression α such that $L(A) = L[\alpha]$.**

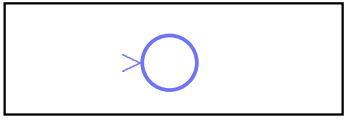
In other words:

Regular languages are precisely those languages that are accepted by finite automata.

From languages to automata

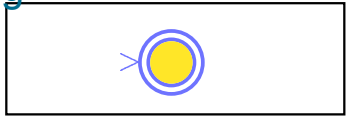
The procedure which converts any regular language $L[\alpha]$ to an NFA A such that $L(A) = L[\alpha]$ operates along the inductive definition of the regular expression α:

- $\alpha = \emptyset$. Then $L[\alpha] = \emptyset$. A :

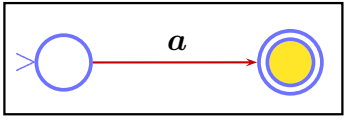


the empty word ε must be accepting

- $\alpha = \varepsilon$. Then $L[\alpha] = \{\varepsilon\}$. A :

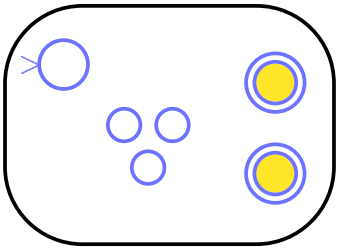


- $\alpha = a$. Then $L[\alpha] = \{a\}$. A :

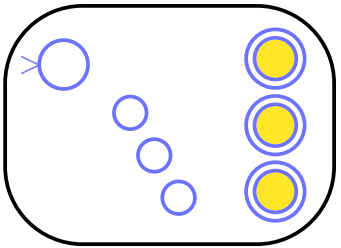


Automaton accepting $L[\alpha \cup \beta]$

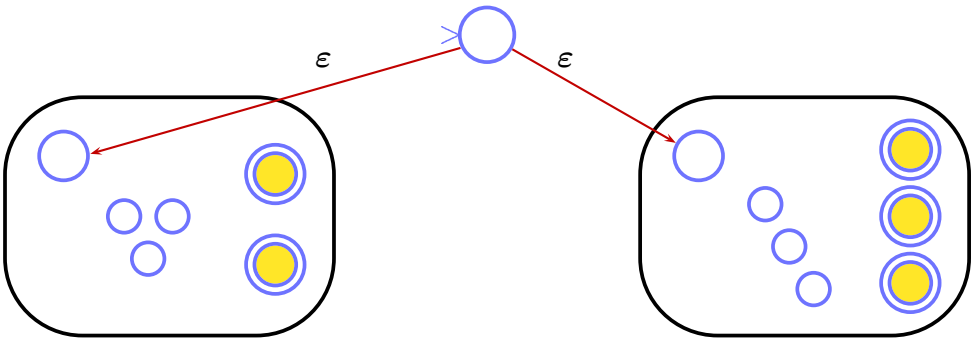
A_1: accepts $L[\alpha]$



A_2: accepts $L[\beta]$

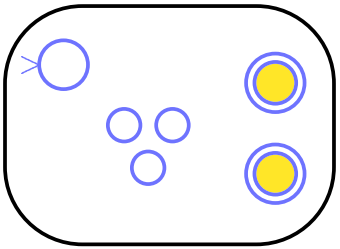


A: accepts $L[\alpha \cup \beta]$

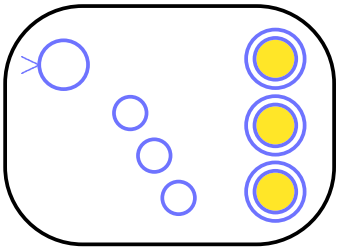


Automaton accepting $L[\alpha\beta]$

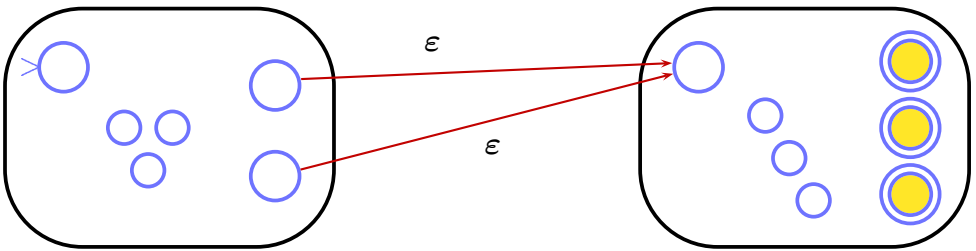
A_1: accepts $L[\alpha]$



A_2: accepts $L[\beta]$

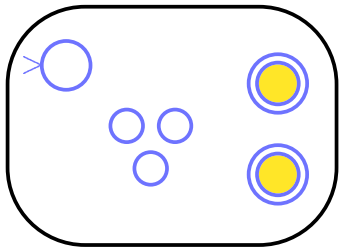


A: accepts $L[\alpha\beta]$

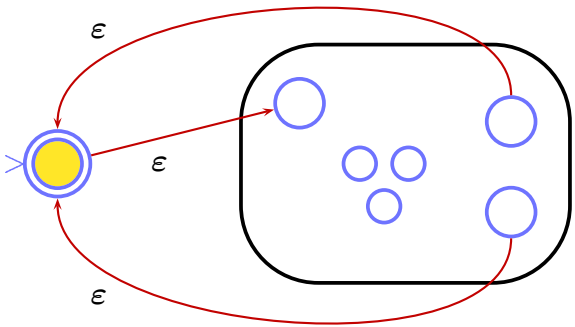


Automaton accepting $L[\alpha^*]$ iteration

A_1: accepts $L[\alpha]$



A: accepts $L[\alpha^*]$



in the coursework, there will be a task to covert a regular expression into an automaton

Example: how the procedure works

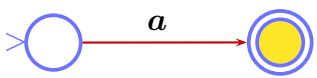
We apply the procedure to the regular expression

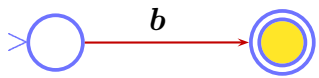
$$((a \cup ab)^* ba)^*$$ this will be in coursework

We construct step by step (going 'inside out') an NFA A such that

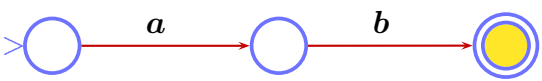
$$L(A) = L[((a \cup ab)^* ba)^*]$$

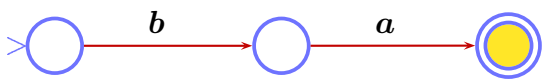
Step 0: automata accepting $L[a]$ and $L[b]$





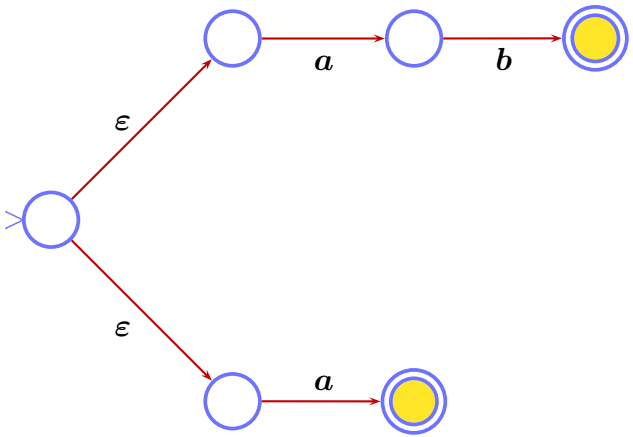
Step 1: automata accepting $L[ab]$ and $L[ba]$





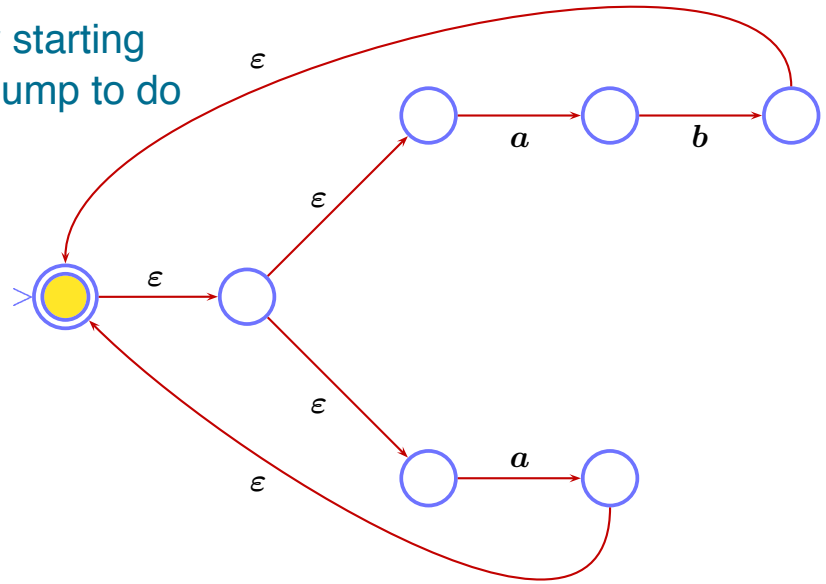
Step 2: automaton accepting $L[a \cup ab]$

use ε-jump to perform union U

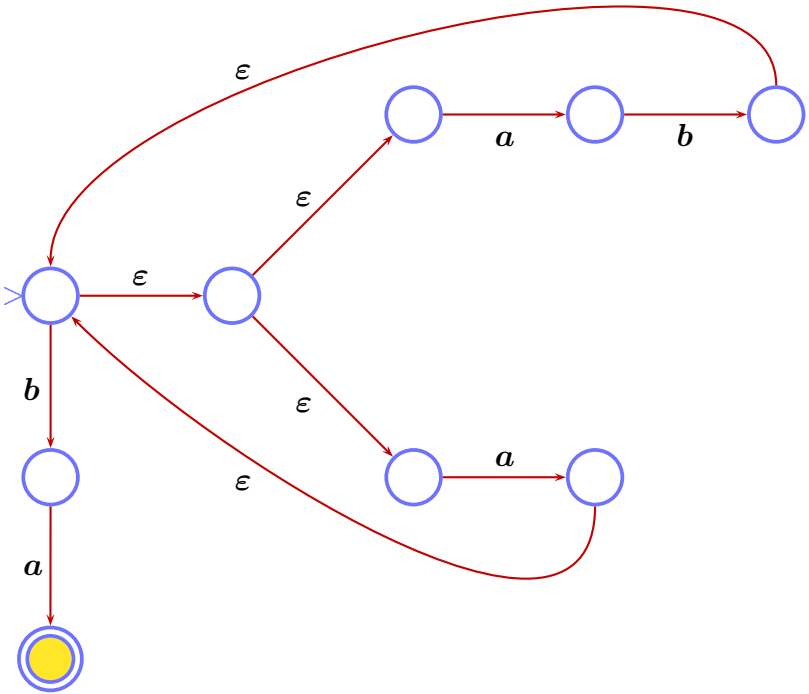


Step 3: automaton accepting $L[(a \cup ab)^*]$

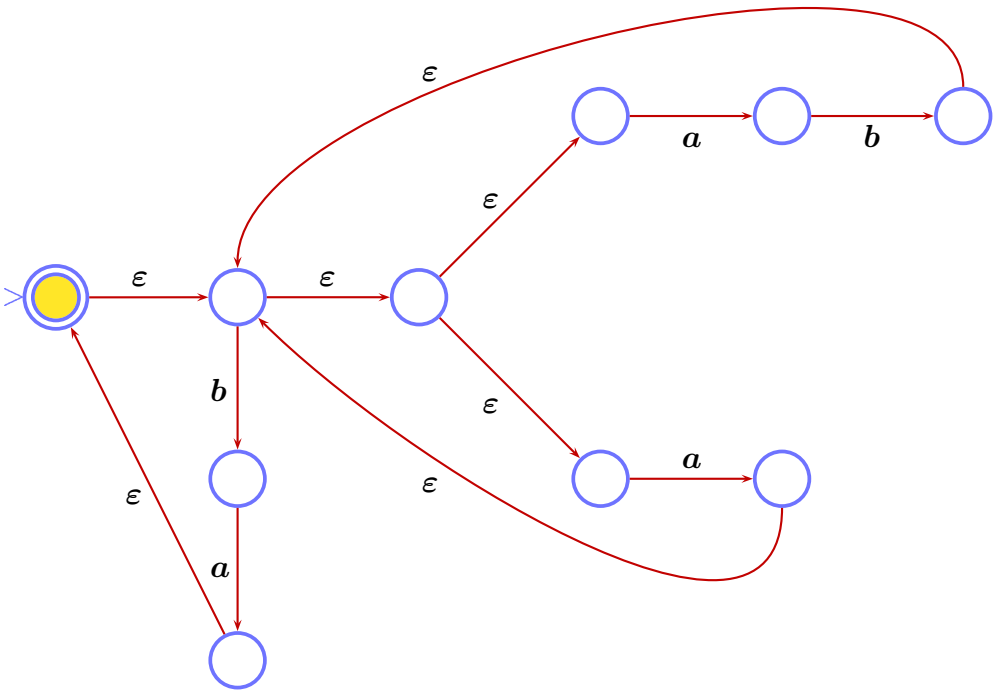
for *:
introduce a new starting state, and use ε-jump to do it



Step 4: automaton accepting $L[(a \cup ab)^*ba]$

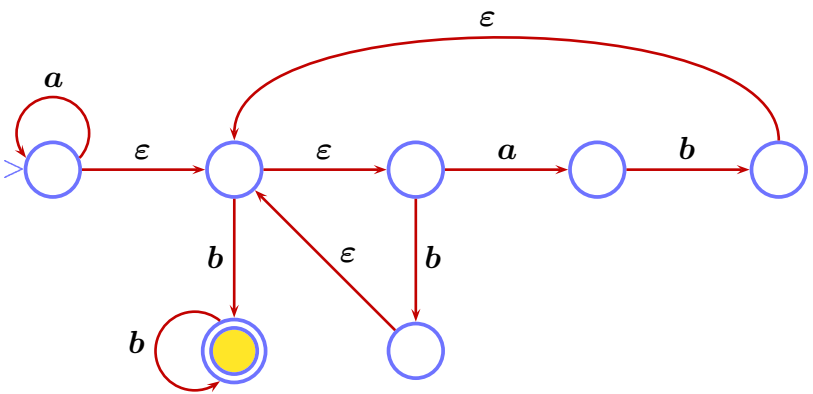


Final step of the procedure



Another example

Automaton accepting $L[a^*(b \cup ab)^*bb^*]$:



Finite automata: summary

- Finite automata may be regarded as programs that use **fixed amounts of memory** (represented by states) regardless of the input.
- Finite automata can be used as **recognition devices:** they accept certain inputs and reject others.
- **Nondeterminism** does not increase the computational power of finite automata, but nondeterministic automata are easier to design than **deterministic** ones.
- The languages accepted by finite automata are precisely the **regular** ones.

Nonregular languages

Finite automata are theoretical models for programs that use
a **constant amount** of memory **regardless** of the input

$\rightsquigarrow$ there should be languages which are **not regular**

Example: Is the following language over the alphabet $\{a, b\}$ regular?

L = all strings starting with a string of a's followed by an equal-length string of b's
$= \{a^n b^n \mid n = 0, 1, 2, \dots\}$ ($a^n b^n$ is **not** a regular expression)

It **does not** seem so: in coursework: is this a regular language? why?

- to recognise strings in L, we may have to store the entire prefix a^n before the first b shows up
- the length of such prefix depends on n $\rightsquigarrow$ constant memory is not enough

? Is it really the case? (the argument above is 'handwaving' there may be other ways to compare the numbers of a and b)

? If it is the case, 'better' models of computation are needed

assume that a language is regular -> prove that it's irregular by using reductio

DFA computations on 'long' inputs

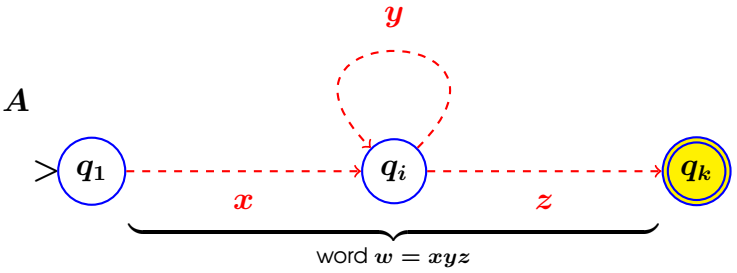
- Given some regular language L with **infinitely many** words,
  take a DFA A that accepts L; let n be the number of states in A
- Consider the computation of A on input $w \in L$ containing $k > n$ symbols:

$$(q_1, w), (q_2, w_2), \ldots, (q_{k-1}, w_{k-1}), (q_k, \varepsilon)$$

pigeonhole principle
(here: 10 pigeons in 9 pigeonholes)

As $n < k$, the states $q_1, q_2, \ldots, q_{k-1}, q_k$ **cannot** be all distinct

so A must contain a **loop**

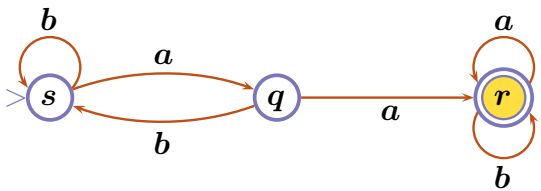


word w can be represented as
$w = xyz$

but then $xy^m z$ is also accepted by A, for any $m \geq 0$

(pumping the middle string y)

Example



Find x, y and z (as on the previous page) for the following input words w:

- $w = ababaa$

  (for example, $x = \varepsilon$, $y = ab$, $z = abaa$)

- $w = aabbb$

  (for example, $x = aa$, $y = b$, $z = bb$)

Pumping Lemma (not examinable)

Let L be an infinite regular language over an alphabet Σ.

Then there is a number $n > 0$ such that, for any $w \in L$ of length $\geq n$,

there exist strings $x, y, z \in \Sigma^*$ such that

- $|xy| \leq n$
- $y \neq \varepsilon$
- $xy^m z \in L$, for any $m \geq 0$

Pumping Lemma is used to show that a given language L

(such as $L = \{a^k b^k \mid k = 0, 1, 2, \ldots\}$)

is **not** regular

If L does not satisfy the 'pumping property,' then L **cannot** be regular

$L = \{a^n b^n \mid n \geq 0\}$ is not regular

Suppose to the contrary that L is regular.

Take the number $n > 0$ provided by the Pumping Lemma and consider $w = a^n b^n$, which belongs to L

By Pumping Lemma, we can represent w as $w = xyz$ with $|xy| \leq n$; but then x and y may only be **strings of a's**

By Pumping Lemma, the word $xy^{n+1}z$ is also in L;

however, $xy^{n+1}z$ contains more a's than b's, and so **cannot** be in L

This contradiction shows: our assumption that L is regular **is not correct**

Thus, the language L is not regular Q.E.D. (*quod erat demonstrandum*)

Exercise: Is the language $L = \{a^{2^n} \mid n \geq 0\}$ regular?