Linguaggi formali

Let's start from the beginning

- A program is written in a programming language
- Every programming language (as every language in general) needs to obey its own rules
- We need to formally define languages…

Reference books

Intoduction to Automata Theory, Languages, And Computation.
Hopcroft, Motwani, Ullman

Fondamenti dell'Informatica. Linguaggi formali, calcolabilita' e complessita'.
Dovier, Giacobazzi
Bollati Boringhieri

Strings

- An alphabet is a finite set of symbols
- Examples

Σ_1 = {a, b, c, d, …, z}: the set of letters in Italian

Σ_2 = {0, 1}: the set of binary digits

Σ_3 = {(,)}: the set of open and closed brackets

A string over alphabet Σ is a finite sequence of symbols in Σ.

- Examples

abfbz is a string over Σ1 = {a, b, c, d, …, z}

11011 is a string over Σ2 = {0, 1}

))()(() is a string over Σ3 = {(,)}

> The empty string is a string having no symbol, denoted by ε.

Strings

- The length of a string x is the number of symbols contained in the string x, denoted by |x|.

- Examples

  |abfbz|=5

  |110010|=6

  |))()(()|=7

  |ε|=0

Strings

- The concatenation of two strings x and y is a string xy,
  i.e., x is followed by y.
  it is an associative operation that admits the neutral element ε

- s is a substring of x if there exist strings y and z
  such that x = ysz.

> Example:
> the prefixes of abc are : ε, a, ab, abc

- In particular,
  when x = sz (substring with y=ε), s is called a prefix of x;
  when x = ys (substring with z=ε), s is called a suffix of x;

ε is a prefix and a suffix of ε and of all strings

Power of an alphabet

- We need to denote the set of all strings over Σ of a given length

Σ^n denotes the strings of length n whose symbols are in Σ

If Σ={0,1}

Σ^0 = {ε}

Σ^1 = Σ = {0,1}

Σ^2 = {00,01,11,10}

Σ^3 = {000,001,010,011, 100,101,110,111}

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \ldots = \bigcup_{i>0} \Sigma^i \qquad \Sigma^* = \{\epsilon\} \cup \Sigma^+$$

Σ^+ = {0,1,00,01,11,10,000,001,010,011, 100,101,110,111….}

Languages

A language is a set of strings over an alphabet:

$L \subseteq \Sigma^*$ is a language over Σ

Examples

L_1 = The set of all strings over Σ_1 that contain the substring "fool"

L_2 = The set of all strings over Σ_2 that are divisible by 7

= {7, 14, 21, …}

L_3 = The set of all strings over Σ_3 where every (is followed by 2 occurrences of)

= {),)),)()), …}

Other examples of Languages

L_4 = The set of binary numbers whose value is prime

{ 10,11,101,111,1011,1101,…}

L_5 = The set of legal English words over the English alphabet

L_6 = The set of legal C programs over the strings of characters

Languages

- The following are operations on sets and hence also on languages.

Union: A U B

Intersection: $A \cap B$

Difference: $A \setminus B$ ($A - B$ when $B \subseteq A$)

Complement: $A = \Sigma^* - A$ where Σ^* is the set of all strings on alphabet Σ.

Concatenation: $AB = \{ab \mid a \in A, b \in B\}$

Example: $\{0, 1\}\{1, 2\} = \{01, 02, 11, 12\}$.

Kleene Closure

Kleene closure: $$A^* = \bigcup_{i=0}^{\infty} A^i$$

- Notation: $$A^+ = \bigcup_{i=1}^{\infty} A^i$$

More example of Languages

Examples:

- The set of strings with n 1's followed by n 0's
  {ε, 01, 0011, 000111, . . .}

- The set of strings with an equal number of 0's and 1's
  {ε, 01, 10, 0011, 0101, 1001, . . .}

- The empty language ∅

- The language {ε} consisting of the empty string only

Remember ∅ $\neq$ {ε}

Problems

- Does the string w belong to the language L?

  Example: $11101 \in L_4$?

We want to define a procedure to decide it!

We can try to generate all words belonging to L_4

We can try to recognise when a word belongs to L_4

Generating a language: Grammars

Starting from a particular initial symbol, using the rewriting rules of the productions,

we generate the set of strings belonging to the language

Grammars I

We define a Grammar $G=(\Sigma, N, S, P)$ where :

- Σ is the alphabet, a set of symbols (called terminals)
- N is the set of nonterminals
- $S \in N$ is the starting symbol
- P is the set of productions, each of the form

$$U \to V$$

where $U \in (\Sigma \cup N)^{+}$ and $V \in (\Sigma \cup N)^{*}$.

Grammars II

$G = (\Sigma, N, S, P)$

A string $w \in \Sigma$ is generated by G if there is a derivation of w using P, starting from the starting symbol S.

$G = (\{a\}, \{S\}, S, P)$

$S \rightarrow \varepsilon$

$S \rightarrow a$

$S \rightarrow aS$

A language generated by grammar G is denoted $L(G)$ and it is the set of strings derived using G.

Grammar Example

We want to describe L1 the language of strings with an even number of 1's

L1 can be generated by a grammar ({0,1},{S,T},S,P) with P equal to

$S \rightarrow \varepsilon$
$S \rightarrow 0S$
$S \rightarrow 1T$
$T \rightarrow 0T$
$T \rightarrow 1S$

A string belongs to L1 iff it can be generated by the grammar

Grammar Example

Does the string 01010 belong to L1?

We need to find a derivation

$S \rightarrow \varepsilon \mid 0S \mid 1T$

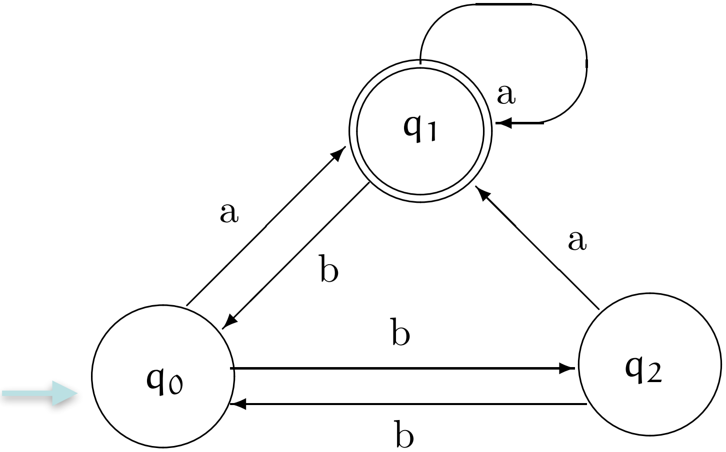
$T \rightarrow 0T \mid 1S$

S

Recognising a language: Automata

- A finite state automaton is finite state machine with an input of discrete values.
- The state machine consumes the input and possibly moves to a different state.
- The system may be in a state among a finite set of possible states. Being in a state allows him to keep track of previous history.

input: bbaa



Back to our Problems

- Does the string w belong to the language L?

We have two ways to answer this question

- Which is the computational complexity necessary to answer to the previous question ?

It depends on the complexity of the language!!

Grammars and Languages

Restrictions on productions give different types of grammars :

- Regular (type 3)
- Context-free (type 2)
- Context-sensitive (type 1)
- Phrase-structure (type 0)

$$U \rightarrow V$$

where $U \in (\Sigma \cup N)^+$ and $V \in (\Sigma \cup N)^*$.

For context-free, e.g., $U \in N$

No restrictions for phrase-structure

A language is of type i iff it admits a grammar of type i (which describes it)

P: decidable in polynomial time

PSPACE: decidable in polynomial space (at least as hard as NP-complete)

U: undecidable

Complexity of Languages Problems

	Regular Grammar Type 3	Context Free Grammar Type 2	Context Sensitive Grammar Type 1	Unrestricted Grammar Type 0
Is $W \in L(G)$?	P	P	PSPACE	U
Is L(G) empty?	P	P	U	U
Is $L(G1) \equiv L(G2)$?	PSPACE	U	U	U

Regular languages

All the following ways to represent regular languages are equivalent:

- Regular grammars (RG, type 3)
- Deterministic finite automata (DFA)
- Non-deterministic finite automata (NFA)
- Non-deterministic finite automata with ε transitions (ε-NFA)
- Regular expressions (RE)

Regular Grammars

A Right (or, analogously, Left) Regular Grammar is a generative grammar, where

- every production has the form A-> aB| a
- only for the starting symbol S, we can have S→ ε

every non terminal symbol B is always preceded by a terminal one.

Example

G=({a,b}, {S,B},S,P) where productions P are:

S-> aS|aB

B->bB|b

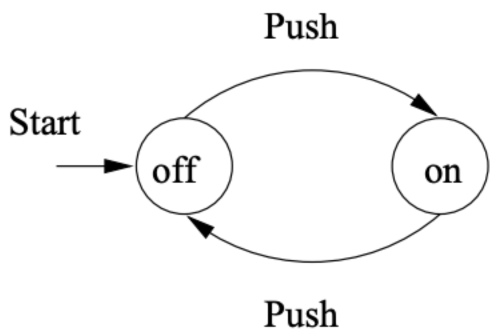
aaabb $\in$ L(G)

S

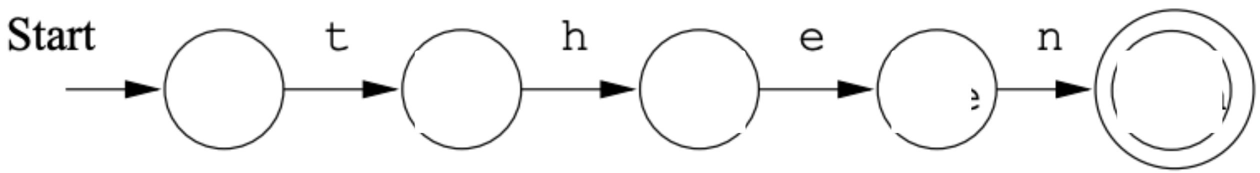
L(G)={ $a^n b^m$| n,m>0}

Deterministic Finite Automata

The states of a switch:



An automaton recognising the keyword then:



Deterministic Finite Automata

A deterministic finite automaton (DFA) $(Q, \Sigma, \delta, qo, F)$

Q a finite set of states

Σ a finite set Σ of symbols

$\delta : Q \times \Sigma \rightarrow Q$ a transition function that takes as argument a state and a symbol and returns one state

qo the starting state

$F \subseteq Q$ the set of final or accepting states

Deterministic Finite Automata

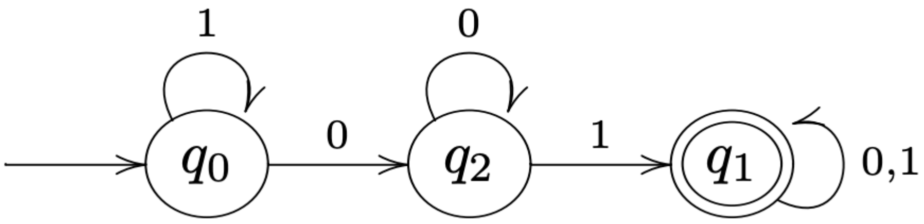
How to represent a DFA? With a transition table

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

-> indicates the starting state

* indicates the final states

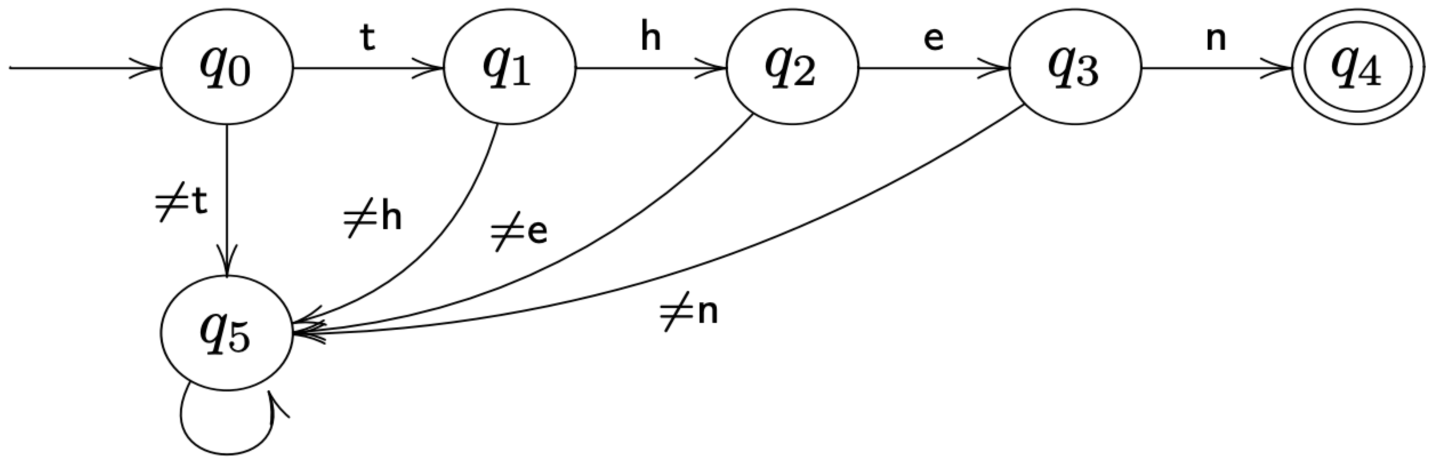
This defines the following transition diagram



Deterministic Finite Automata

When does an automaton accept a word?

It reads a word and accept it if it stops in an accepting state



here Q= $\{q_0, q_1, q_2, q_3, q_4, q_5\}$ F= $\{q_4\}$

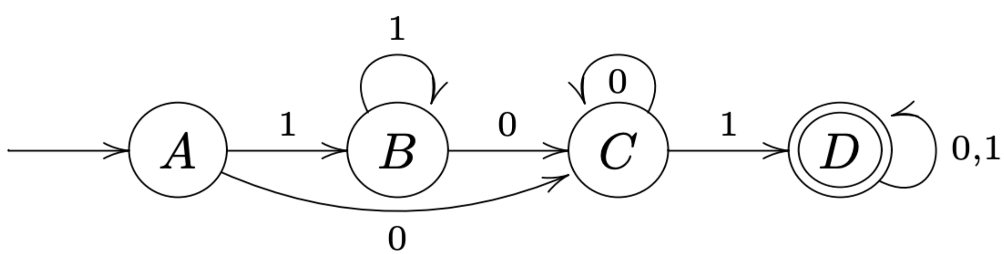
Only the word then is accepted

How DFA processes Strings

We build an automaton that accepts string containing the substring 01

$\Sigma=\{0,1\}$
$L=\{x01y \mid x,y \in \Sigma^*\}$

We get



	0	1
$\rightarrow$A	C	B
B	C	B
C	C	D
$*$D	D	D

Extending the transition function to Strings

We define the transitive closure of δ

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow Q$$

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = & q \\ \hat{\delta}(q, wa) & = & \delta(\hat{\delta}(q, w), a) \end{cases}$$

A string x is accepted by M=(Q, Σ, δ ,qo,F) iff $\widehat{\delta}(q_0, x) \in F$

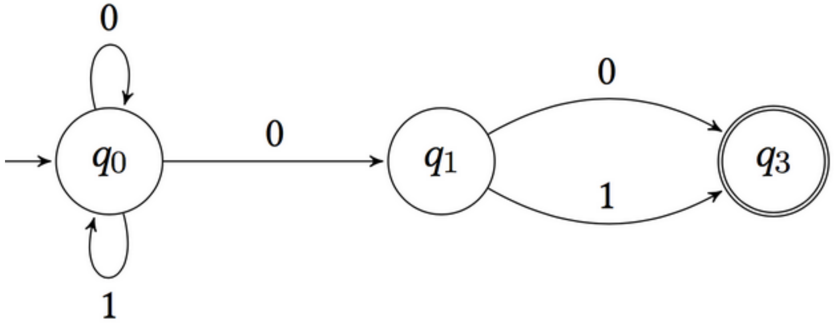
$$L(M) = \{x \in \Sigma^* | \widehat{\delta}(q_0, x) \in F\}$$

Nondeterministic Finite Automata

A nondeterministic finite automaton (NFA) allows more than one transition on the same input symbol.

Formally, a NFA is defined as (Q, Σ, δ, q_0, F) where the only difference is the transition function

$\delta : Q \times \Sigma \rightarrow \wp(Q)$ a transition function that takes as argument a state and a symbol and returns a set of states



Extending the transition function to Strings

We define the transitive closure of δ

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = & \{q\} \\ \hat{\delta}(q, wa) & = & \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a) \end{cases}$$

□

A string x is accepted by M=(Q, Σ, δ ,qo,F) iff $\widehat{\delta}(q_0, x) \cap F \neq \emptyset$

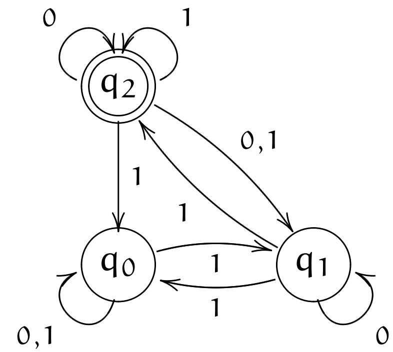
$$L(M) = \{x \in \Sigma^* | \widehat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

NFAs do not expand the class of language that can be accepted !

Example

	0	1
→ q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
* q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

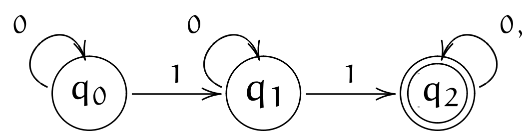
$F = \{q_2\}$.



NFA

L= $\{x \in \{0,1\}^* \mid$ x contains at least 2 occurrences of 1 }

	0	1
→ q_0	q_0	q_1
q_1	q_1	q_2
* q_2	q_2	q_2



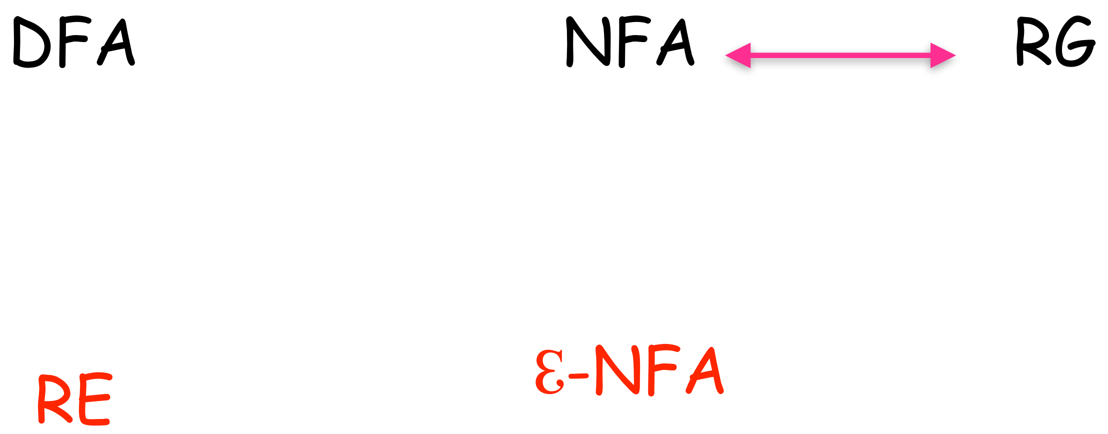
DFA

Different characterisation of Regular Languages

There are different ways to characterise a regular language

- Regular grammars
- Deterministic Finite Automata
- Non Deterministic Finite Automata
- Epsilon Non deterministic Finite Automata
- Regular expression

Roadmap: equivalence between NFA and RG



From Regular Grammars to NFA

Theorem 1.
For each right grammar RG (or left grammar LG), there is a non deterministic finite automaton NFA such that L(RG)=L(NFA).

Construction Algorithm

For a given right grammar RG=(Σ, N, S, P) there is a corresponding NFA=(N U{F}, Σ, δ,S, F') where F is a newly added state and if F'= {F}U{S} if S-> ε belongs to P, F'= {F}, otherwise.

The transition function δ is defined by the following rules.

1) For any A->a belonging to P, with a in Σ, set $\delta(A,a) = F$
2) For any A-> aB belonging to P, with a in Σ and B in N, set $\delta(A,a)=B$

Example

G=({a,b}, {S,B},S,P) where productions P are:

S-> aS|aB

B->bB|b $\qquad$ $L(G)=\{\ a^n b^m \mid n,m>0\}$

From NFA to Regular Grammars

Theorem 2

For each nondet finite automaton NFA, there is one right grammar RG (or left grammar LG) where L(RG)=L(NFA).

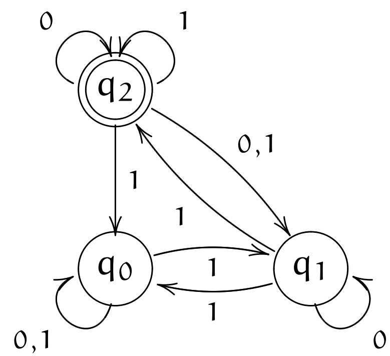
For a given finite automata NFA= ($Q, \Sigma, \delta, q_0, F$), a corresponding right grammar RG=(Σ, Q, q_0', P) can be constructed using the following steps

1) for any $\delta(A,a)=B$ add $A \rightarrow aB$ to P,
2) if B belongs to F add also $A \rightarrow a$ to P;

If q_0 belongs to F then add $q \rightarrow q_0 \mid \varepsilon$ to P and $q_0'=q$ else $q_0'=q_0$.

Example

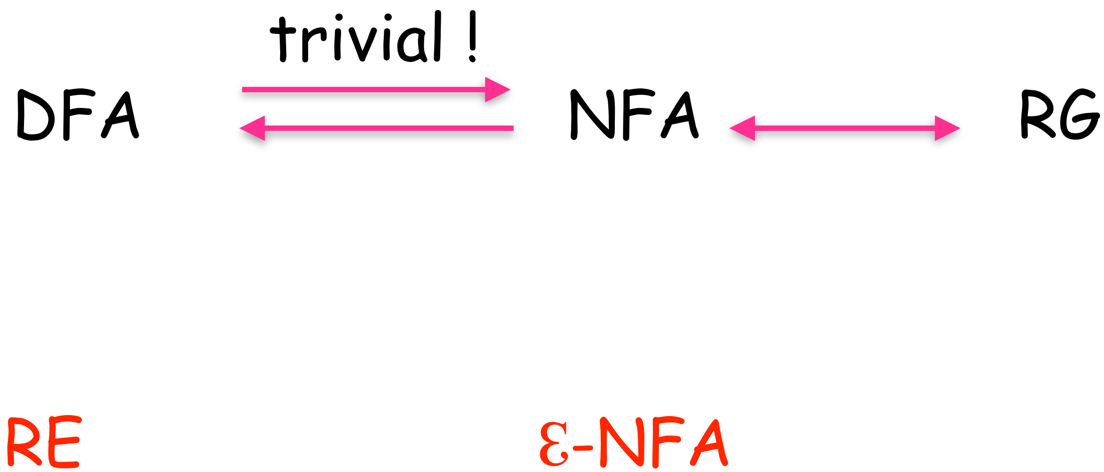
		0	1
→	q_0	$\{q_0\}$	$\{q_0, q_1\}$
	q_1	$\{q_1\}$	$\{q_0, q_2\}$
*	q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



NFA

L = { $x \in \{0,1\}^* \mid x$ cor x contains at least 2 occurrences of 1 }

Roadmap: equivalence between DFA and NFA



From a NFA to a DFA

The NFA are usually easier to "program".

For each NFA N there is a DFA D, such that L (D) = L (N),.

This involves a subset construction.

Given an

NFA N =

$$(Q_N, \Sigma, \delta_N, q_0, F_N)$$

we will build a

DFA D =

$$(Q_D, \Sigma, \delta_D, q_0, F_D)$$

such that

L (D) = L (N)

From NFA to a DFA

$$Q_D = \wp(Q_N),$$

Note that not all these state are necessary, most of them will be unreachable.

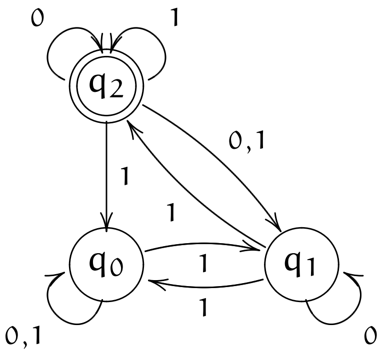
$$\forall P \in \mathcal{P}(Q_N) : \qquad \delta_D(P, a) = \bigcup_{p \in P} \delta_N(p, a)$$

$$F_D = \{P \in \mathcal{P}(Q_N) \mid P \cap F \neq \emptyset\}$$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
* q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



Consider all the subsets $\mathcal{P}(Q_N)$

$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

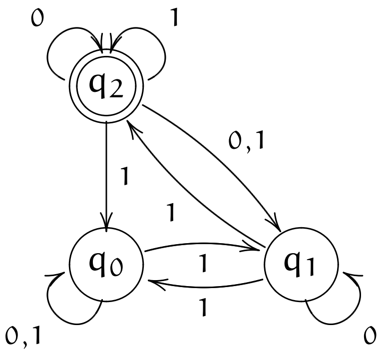
$\{q_0, q_1, q_2\}$

Which ones are final?

Example

NFA

		0	1
	q_0	$\{q_0\}$	$\{q_0, q_1\}$
	q_1	$\{q_1\}$	$\{q_0, q_2\}$
*	q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\mathcal{P}(Q_N)$ $\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

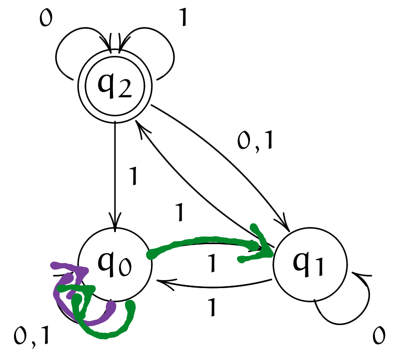
$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
$*q_2$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q_0'	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$

$\mathcal{P}(Q_N)$

$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

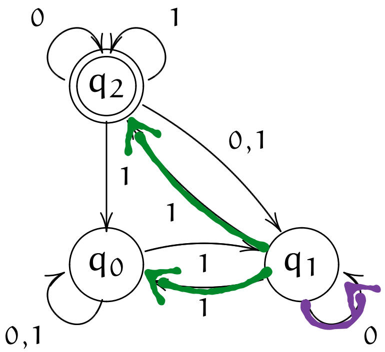
$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
$\star q_2$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\mathcal{P}(Q_N)$

$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

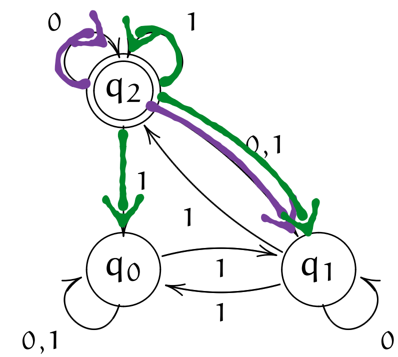
$\{q_0, q_1, q_2\}$

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q'_0	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q'_1	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
* q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\mathcal{P}(Q_N)$

$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

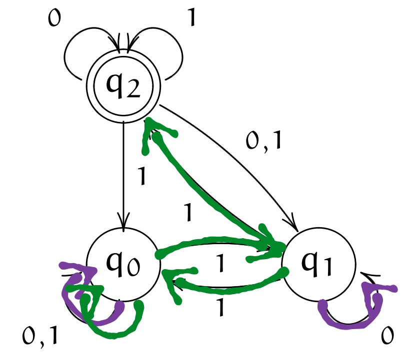
$\{q_0, q_1, q_2\}$

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q_0'	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q_1'	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
* q_2'	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
$*q_2$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

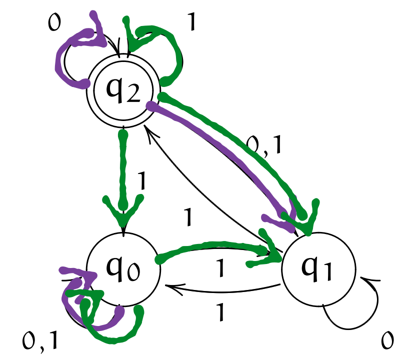
$\{q_0, q_1, q_2\}$

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q_0'	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q_1'	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
$*q_2'$	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q_3'	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
$*q_2$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

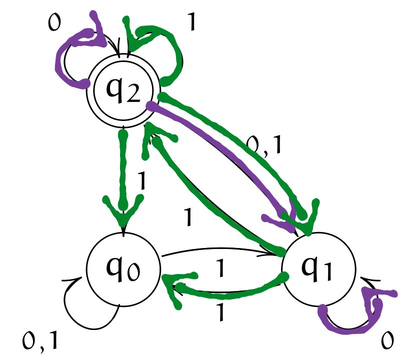
$\{q_0, q_1, q_2\}$

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q'_0	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q'_1	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
$*q'_2$	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q'_3	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$*q'_4$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
$\star\ q_2$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

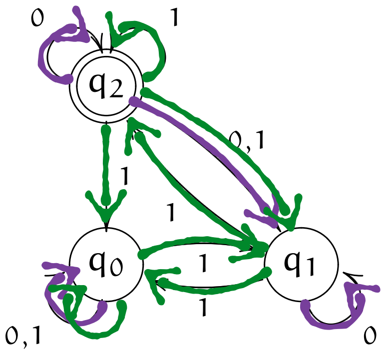
$\{q_0, q_1, q_2\}$

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q_0'	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q_1'	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
$\star\ q_2'$	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q_3'	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$\star\ q_4'$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
$\star\ q_5'$	$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Example

NFA

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
* q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$



$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
q'_0	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q'_1	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
* q'_2	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q'_3	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
* q'_4	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
* q'_5	$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
* q'_6	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Example

NFA

		0	1
	q_0	$\{q_0\}$	$\{q_0, q_1\}$
	q_1	$\{q_1\}$	$\{q_0, q_2\}$
*	q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

$\emptyset$

$\{q_0\}$ $\{q_1\}$ $\{q_2\}$

$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

			0	1
		$\emptyset$	$\emptyset$	$\emptyset$
	q'_0	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
	q'_1	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
*	q'_2	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
	q'_3	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
*	q'_4	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
*	q'_5	$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
*	q'_6	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

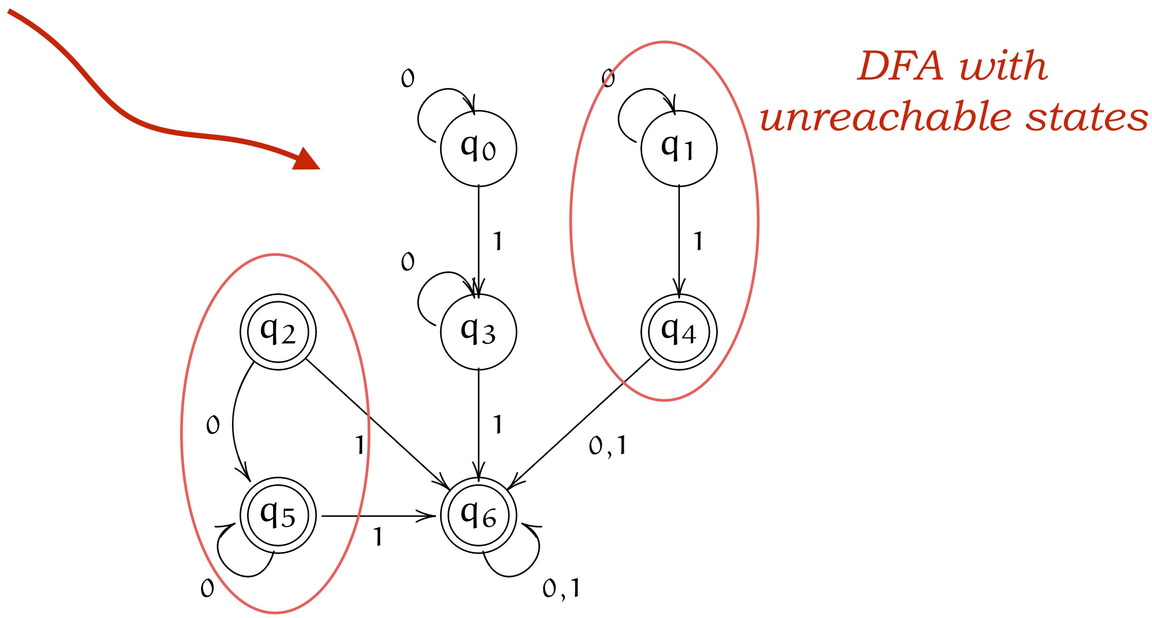
DFA

		0	1
	q_0	q_0	q_3
	q_1	q_1	q_4
*	q_2	q_5	q_6
	q_3	q_3	q_6
*	q_4	q_6	q_6
*	q_5	q_5	q_6
*	q_6	q_6	q_6

Example

DFA

		0	1
	q_0	q_0	q_3
	q_1	q_1	q_4
*	q_2	q_5	q_6
	q_3	q_3	q_6
*	q_4	q_6	q_6
*	q_5	q_5	q_6
*	q_6	q_6	q_6

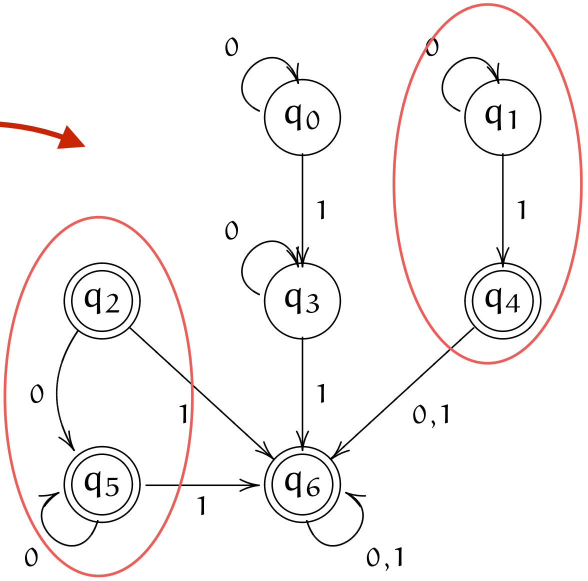


Example

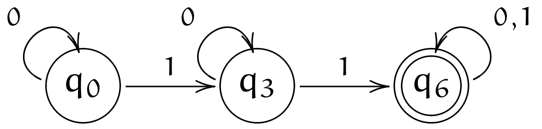
DFA

	0	1
q_0	q_0	q_3
q_1	q_1	q_4
* q_2	q_5	q_6
q_3	q_3	q_6
* q_4	q_6	q_6
* q_5	q_5	q_6
* q_6	q_6	q_6

DFA with unreachable states



minimum DFA



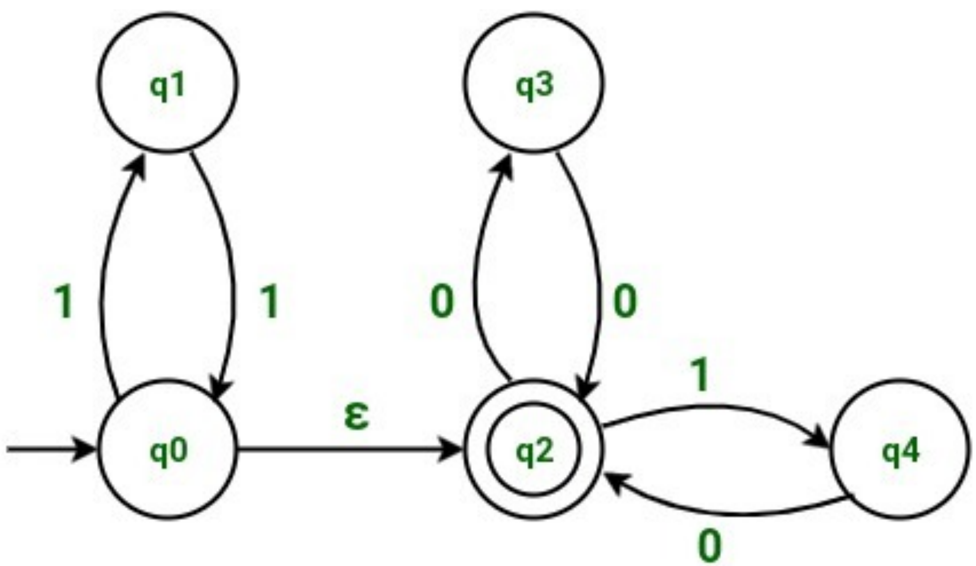
The ε-NFA: NFA with epsilon transitions

- Extension of finite automaton.
- The new feature: we allow transition on ϵ , the empty string.
- An NFA that is allowed to make transition spontanously , without receiving any input symbol.
- As in the case of NFA w.r.t. DFA this new feature does not expand the class of language that can be accepted.

Definition of ε-NFA

A NFA whose transition function can always choose epsilon as input symbol

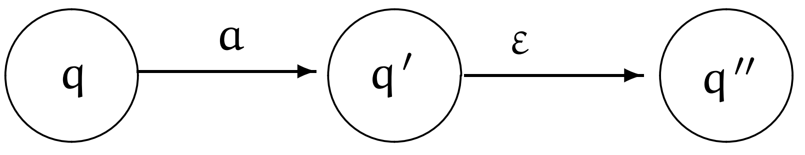
$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \wp(Q)$$



Definition of ε-clousure for extending δ to Strings

We need to define the $\varepsilon\text{-}closure$ that applied to a state gives all the states reachable with ε-transitions

$$\varepsilon\text{-closure}(P) = \bigcup_{p \in P} \varepsilon\text{-closure}(p)$$

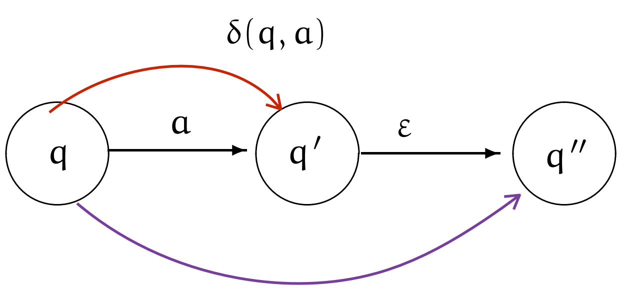


ϵ-closure(q)=$\{q\}$ ϵ-closure(q')=$\{q', q''\}$

The extension of δ to Strings

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$$

$$\begin{cases} \hat{\delta}(q, \varepsilon) = \varepsilon\text{-closure}(q) \\ \hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \varepsilon\text{-closure}(\delta(p, a)) \end{cases}$$

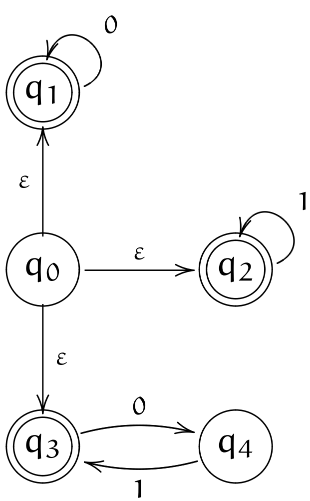


$$\hat{\delta}(q, a) = \bigcup_{p \in \hat{\delta}(q,\varepsilon)} \varepsilon\text{-closure}(\delta(p, a)) = \{q', q''\}$$

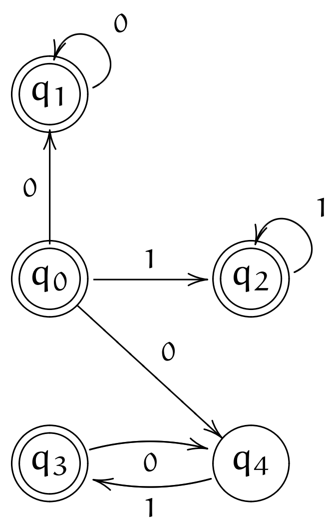
Example

$$L = \{ \; x \mid \exists n \in \mathbb{N}.\; x = 0^n \;\vee\; x = 1^n \;\vee\; x = (01)^n \; \}$$

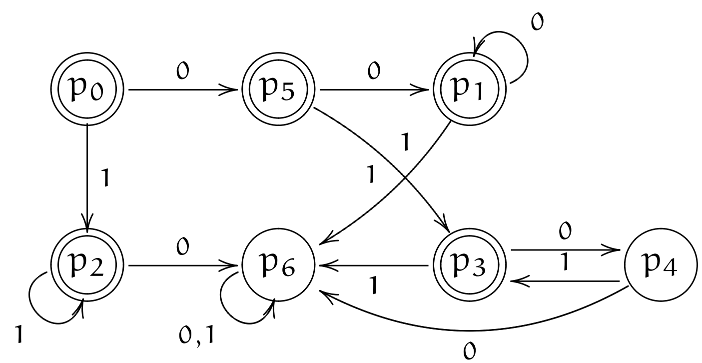
ε-NFA



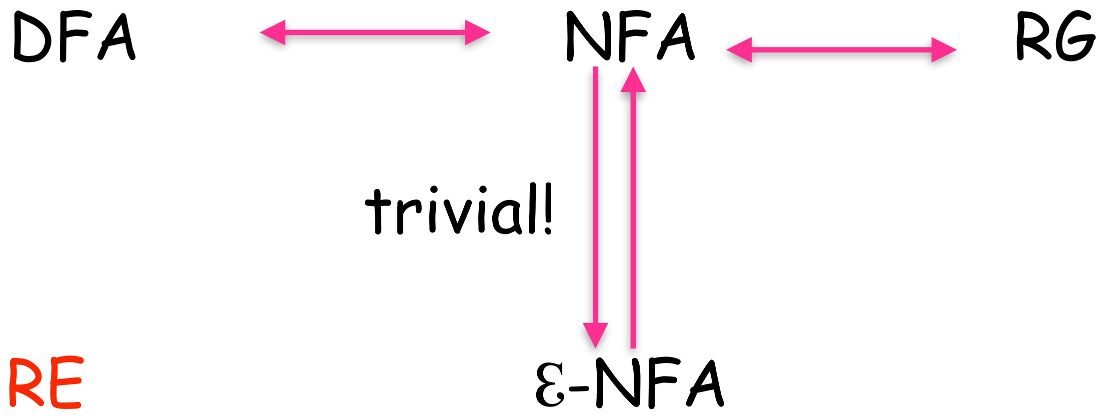
NFA



DFA



Roadmap: equivalence between NFA and ε-NFA



From ε-NFA to NFA

For each ε-NFA E there is a NFA N, such that L (E) = L (N).

Given an

ε-NFA E = $(Q, \Sigma, \delta_E, q_0, F_E)$

we will build a

NFA N = $(Q, \Sigma, \delta_N, q_0, F_N)$

such that

L (E) = L (N)

From ε-NFA to NFA

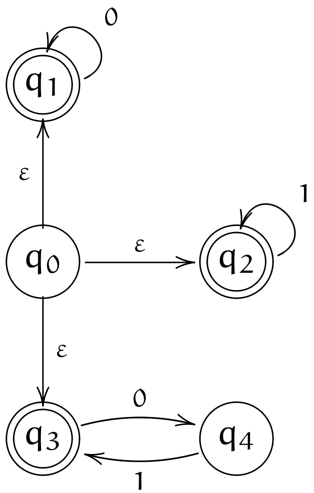
$$\delta_N(q,a) = \widehat{\delta}_E(q,a)$$

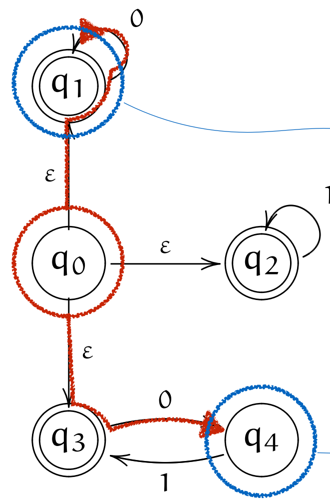
$$F_N = \begin{cases} F_E \cup \{q_0\} & \text{if } \epsilon\text{-closure}(q_0) \cap F_E \neq \emptyset \\ F_E & \text{otherwise} \end{cases}$$

(if a final state can be reached with an epsilon transition from the initial state)

Example

ε-NFA

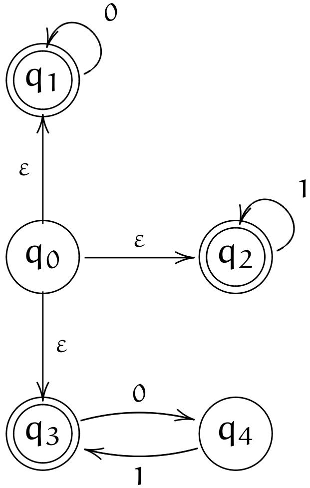


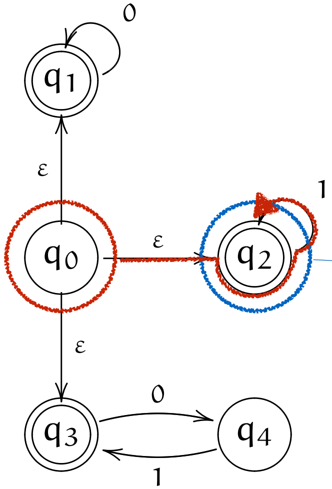


	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Example

ε-NFA

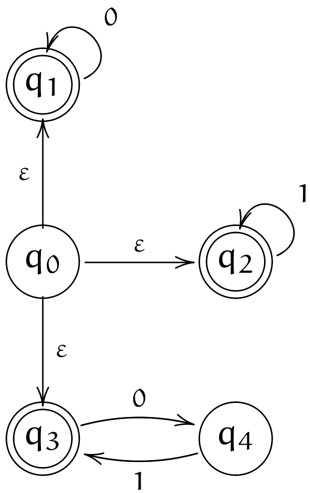


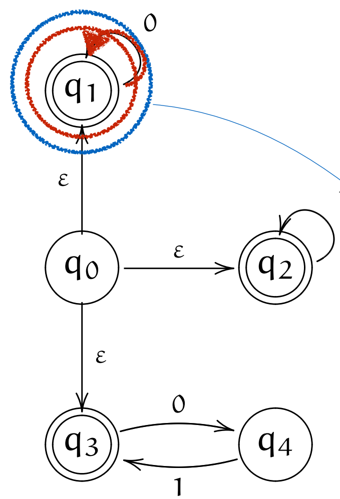


	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Example

ε-NFA

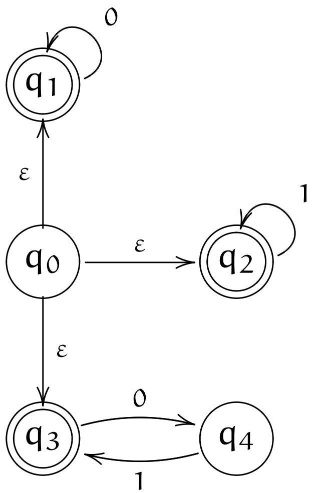


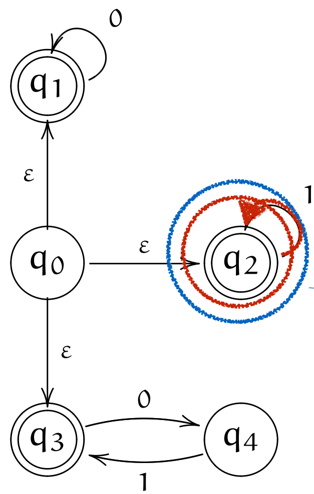


	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Example

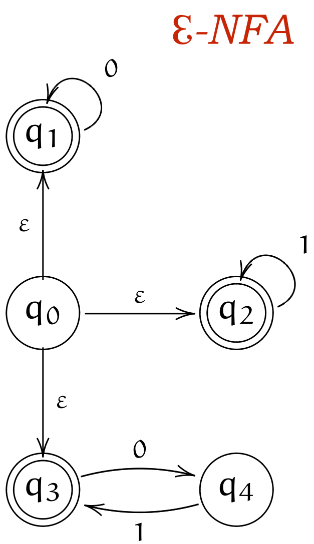
ε-NFA

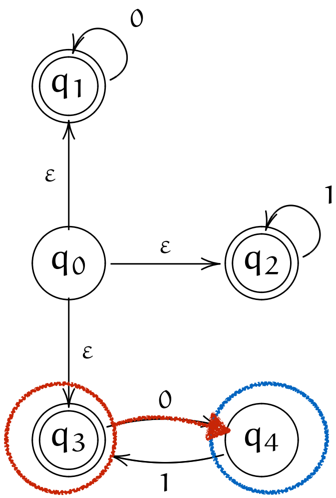




	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Example

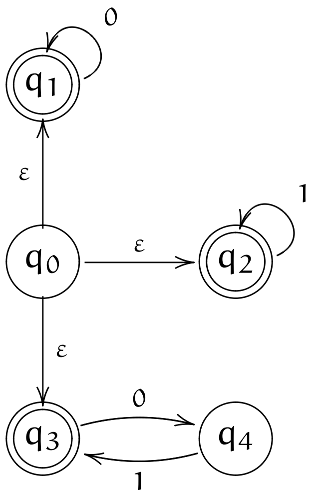


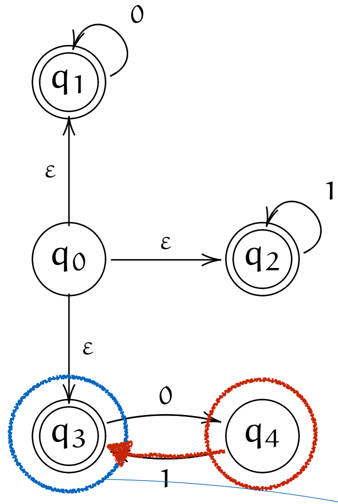


	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Example

ε-NFA

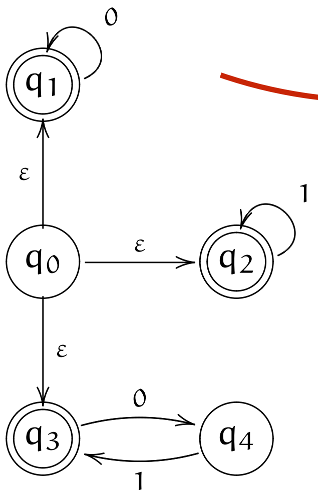




	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Example

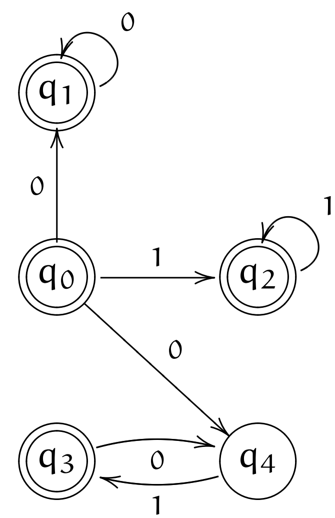
ε-NFA



NFA

	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

NFA



Operations on languages: recap.

Union: A U B

Intersection: $A \cap B$

Difference: $A \setminus B$

Complement: $A = \Sigma^* - A$

Concatenation: $AB = \{ab \mid a \in A,\ b \in B\}$

Kleene Clousure:

$$A^* = \bigcup_{i=0}^{\infty} A^i$$

Regular Expressions

A regular expression denotes a set of strings.

Given a finite alphabet Σ, the following constants are defined as regular expressions:

- ∅ denoting the empty set,
- ε denoting the set {ε},
- a in Σ denoting the set containing only the character {a}

If r and s are regular expression denoting the sets R and S, then,

(r+s),(rs) and r* denotes the set R U S, RS and R*, respectively.

L(r) indicates the language denoted by r

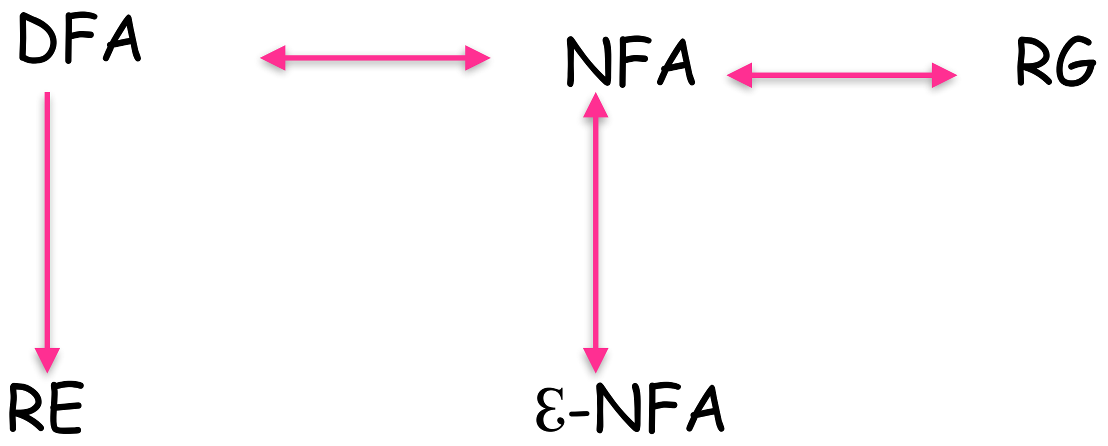
Examples

$$(0^* + 1^* + (01)^*).$$

$$L = \{\ x \mid \exists n \in \mathbb{N}.\ x = 0^n\ \vee\ x = 1^n\ \vee\ x = (01)^n\ \}$$

- a|b* denotes
  {ε, "a", "b", "bb", "bbb", ...}
- (a|b)* denotes
  all the strings formed with "a" and "b"
- ab*(c|ε) denotes
  the set of strings starting with "a", then zero or more "b"s and finally optionally a "c"
- (0|(1(01*0)*1))*
  denotes the set of binary numbers that are multiples of 3

Roadmap



Turning a DFA into a RE

Theorem 3

For each DFA D, there is a regular expression r such that L(D)=L(r).

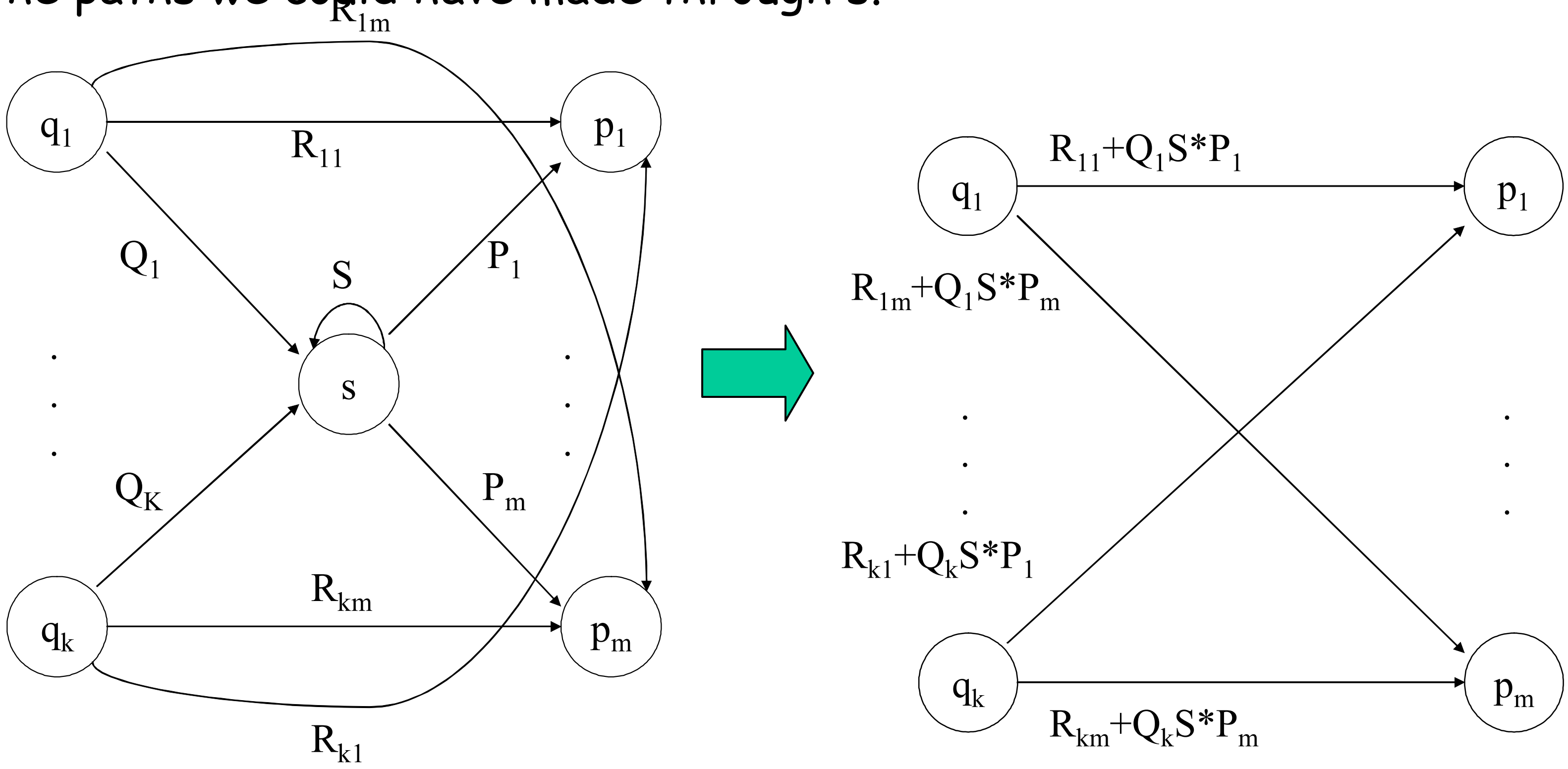
Construction:

- Eliminates states of the automaton and replaces the edges with regular expressions that includes the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE

State Elimination

Note: q and p may be the same state!

- Consider the figure below, which shows a generic state s about to be eliminated. The labels on all edges are regular expressions.
- To remove s, we must make labels from each q_i to p_1 up to p_m that include the paths we could have made through s.



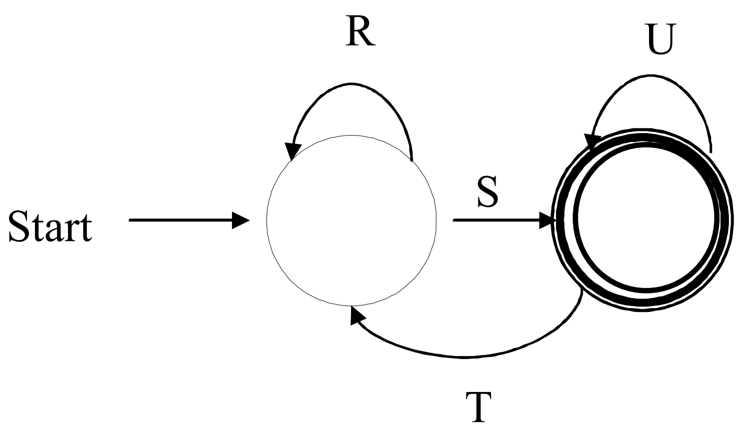
DFA to RE via State Elimination

Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.

- The result will be some (one or more than one) state automaton with a start state and accepting state.

DFA to RE State Elimination (2)

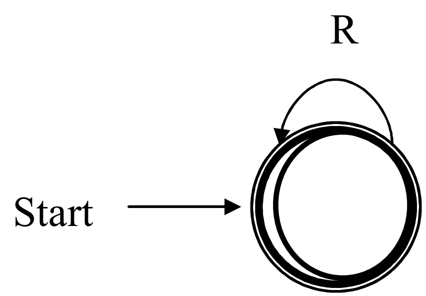
If the two states are different, we will have an automaton that looks like the following:



We can describe this automaton as: (R+SU*T)*SU*

DFA to RE State Elimination (3)

If the start state is also an accepting state, then we must also perform a state elimination from the original automaton that gets rid of every state but the start state. This leaves the following:



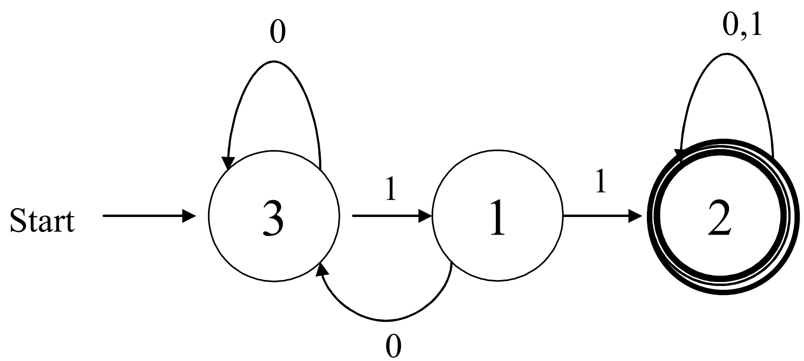
We can describe this automaton as simply R*.

DFA to RE State Elimination (4)

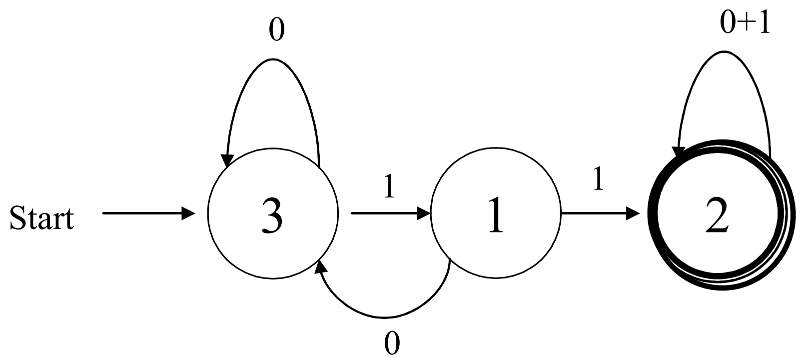
If there are n accepting states, we must repeat the above steps for each accepting states to get n different regular expressions, R_1, R_2, … R_n. For each repeat we turn any other accepting state to non-accepting. The desired regular expression for the automaton is then the union of each of the n regular expressions: $R_1 \cup R_2 \ldots \cup R_N$

DFA→RE Example

- Convert the following to a RE

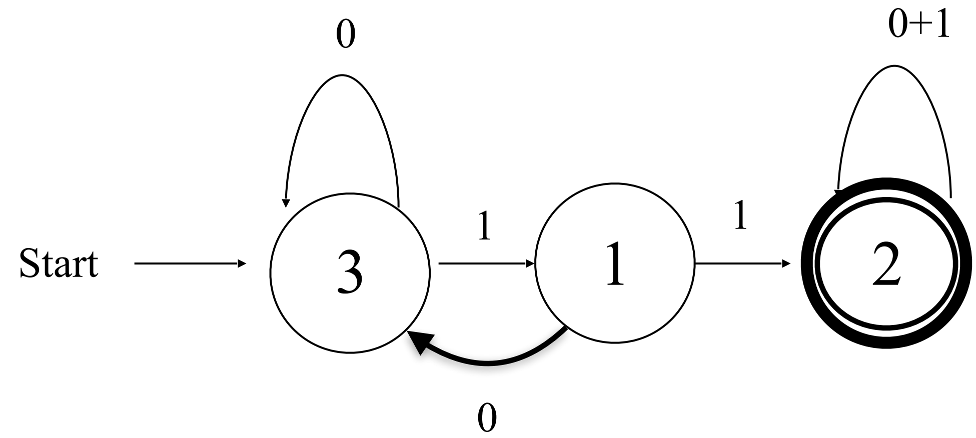


- First convert the edges to RE's:



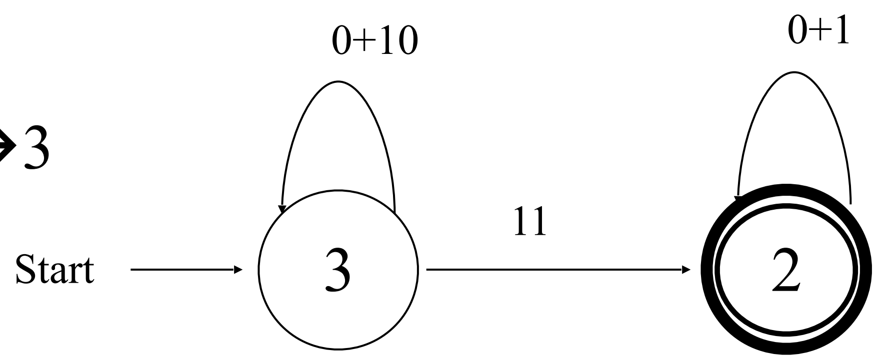
DFA → RE Example (2)

- we want to eliminate State 1:



- obtaining:

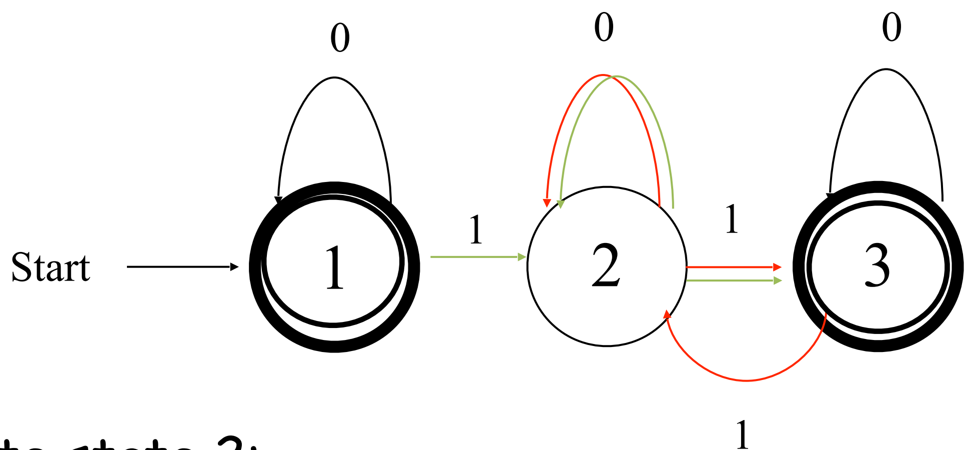
Note edge from 3→3



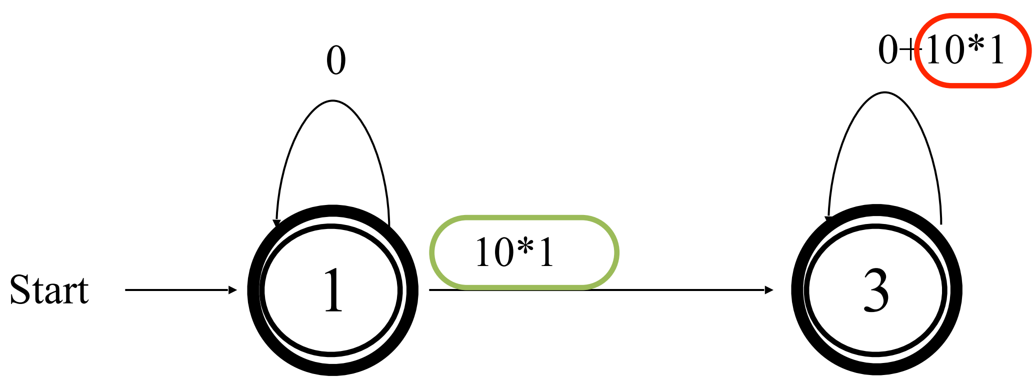
Answer: (0+10)*11(0+1)*

Third Example

- Automata that accepts even number of 1's

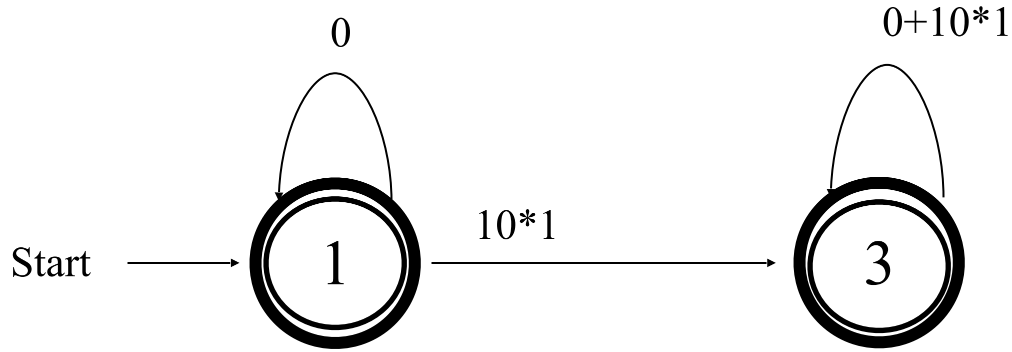


- Eliminate state 2:

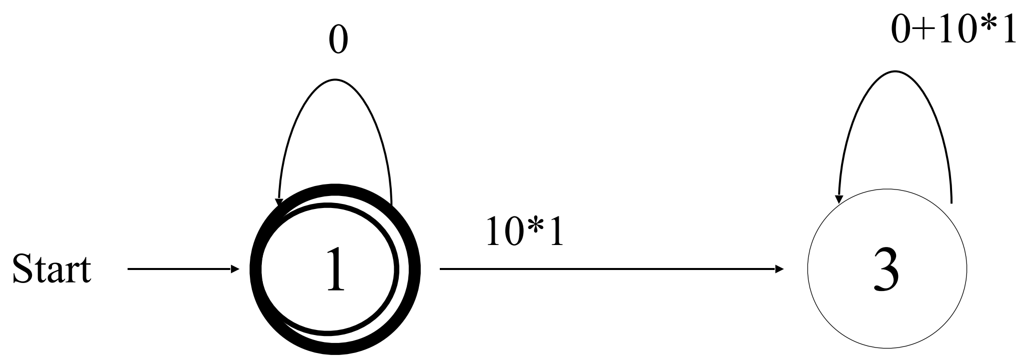


Third Example (2)

- Two accepting states, turn off state 3 first

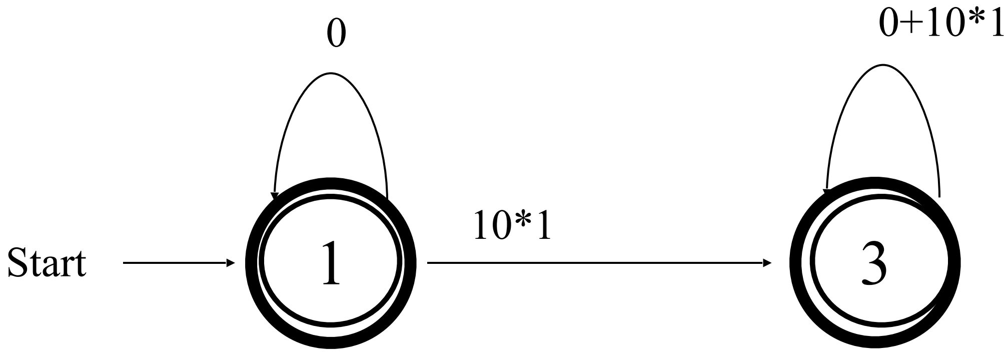


This is just 0*; can ignore going to state 3 since we would "die"

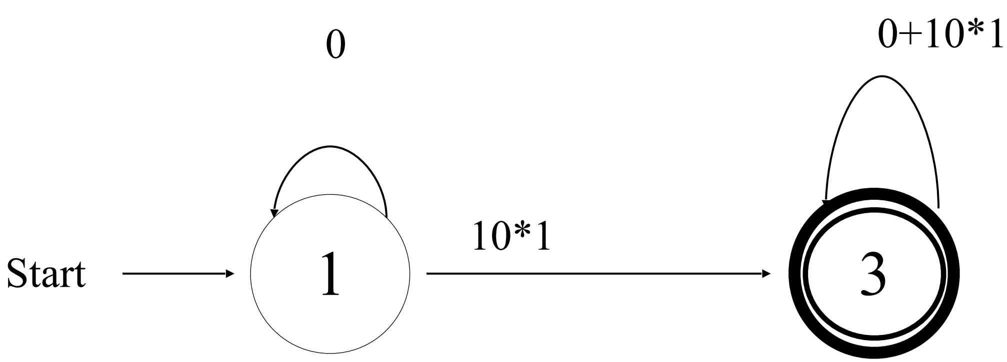


Second Example (3)

- Turn off state 1 second:

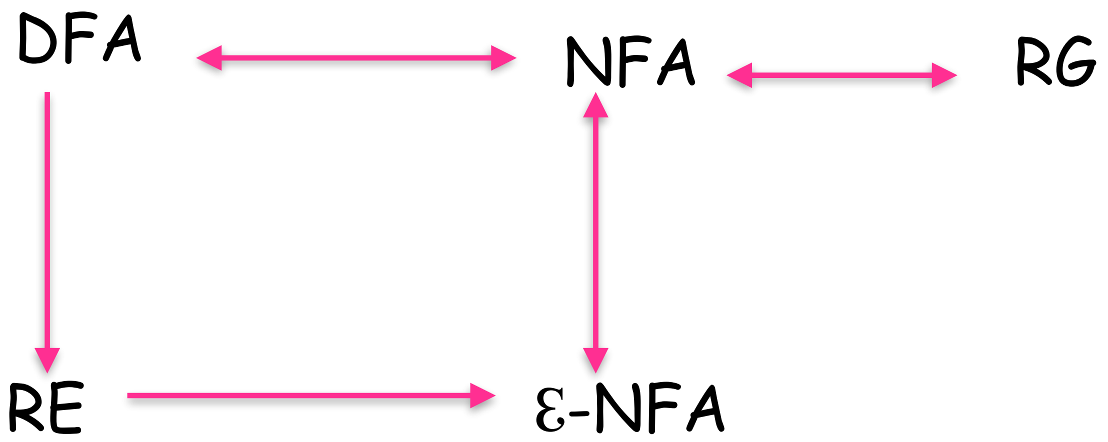


This is just 0*10*1(0+10*1)*



Combine from previous slide to get
0* + 0*10*1(0+10*1)*

Roadmap



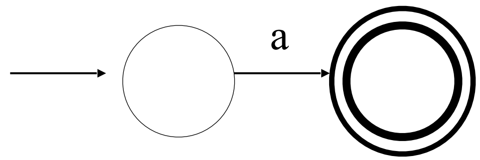
Converting a RE to an Automata

- We can convert a RE to an ε-NFA
  - Inductive construction
  - Start with a simple basis, use that to build more complex parts of the NFA

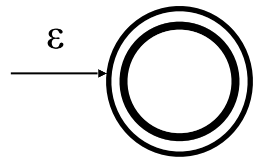
RE to ε-NFA

- Basis:

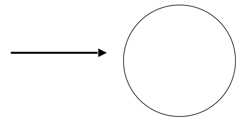
R=a



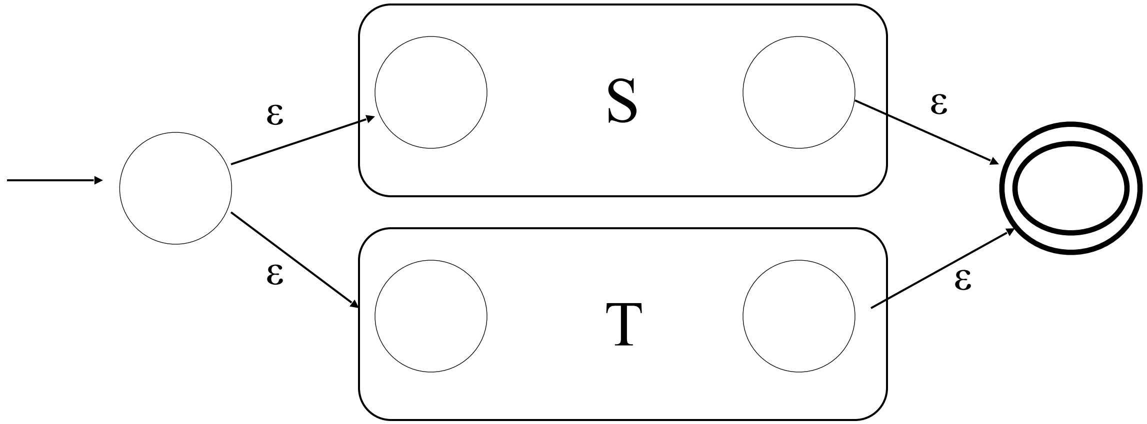
R=ε



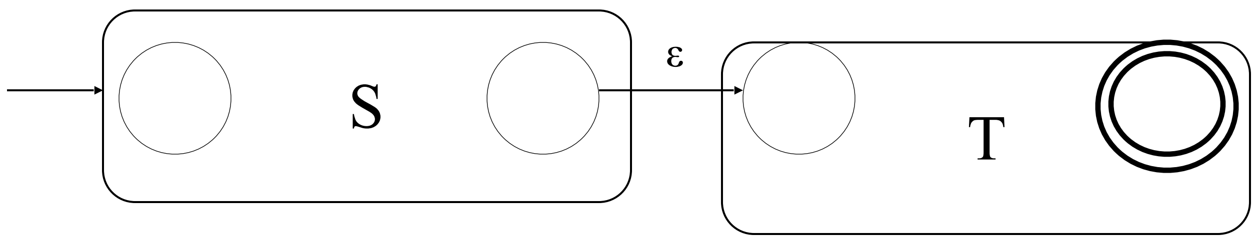
R=Ø



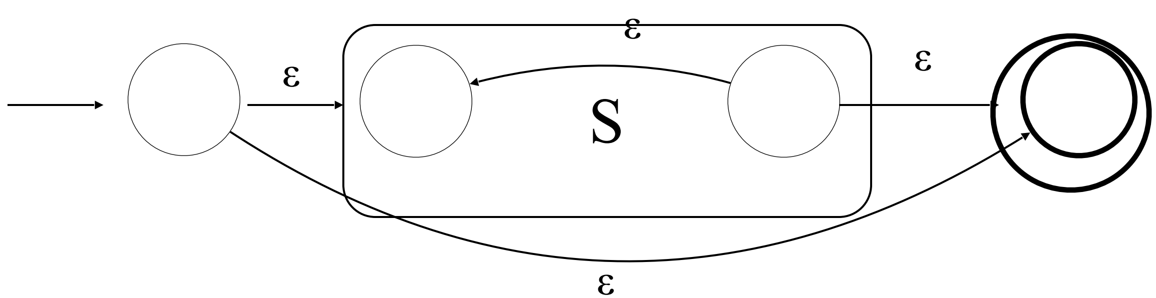
R=S+T



R=ST

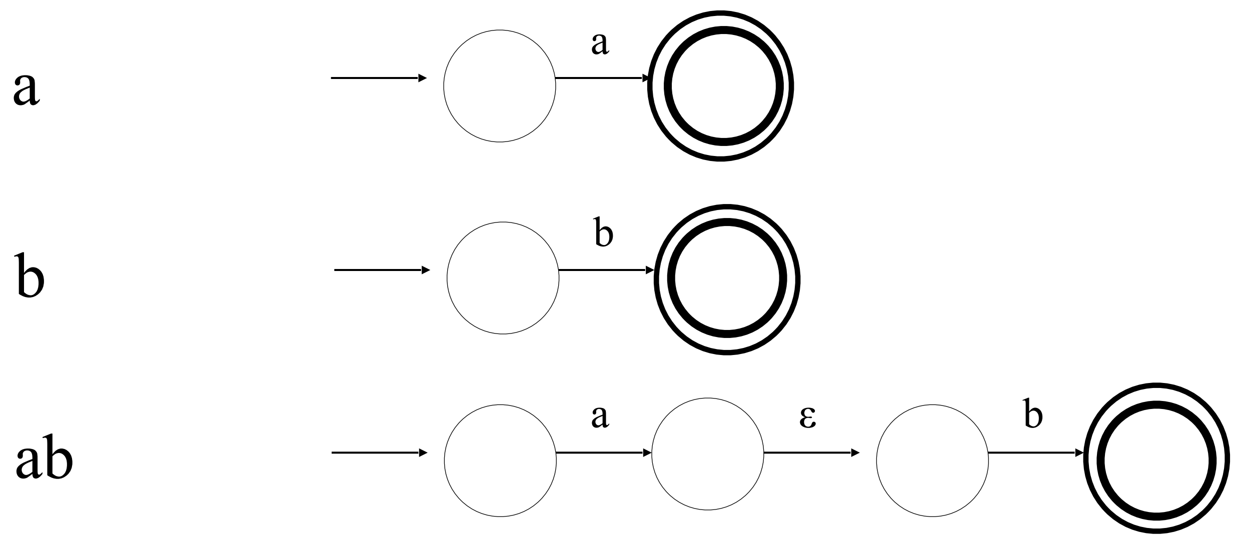


R=S*



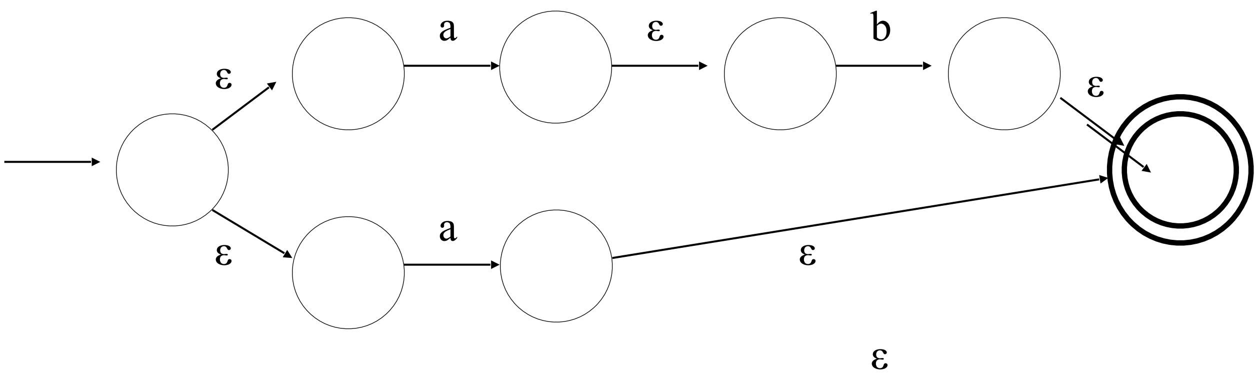
RE to ε-NFA Example

- Convert R= (ab+a)* to an NFA
  - We proceed in stages, starting from simple elements and working our way up

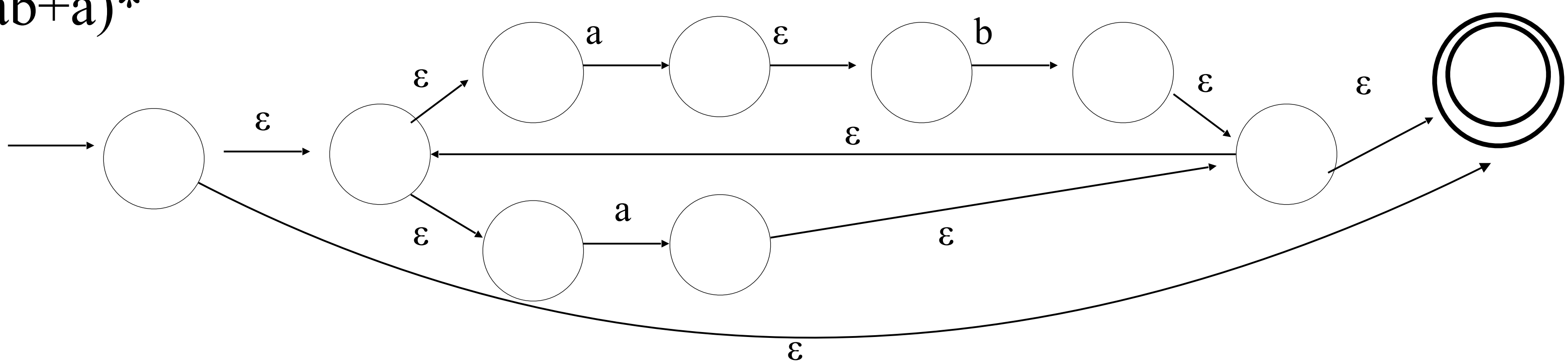


RE to ε-NFA Example (2)

ab+a



(ab+a)*

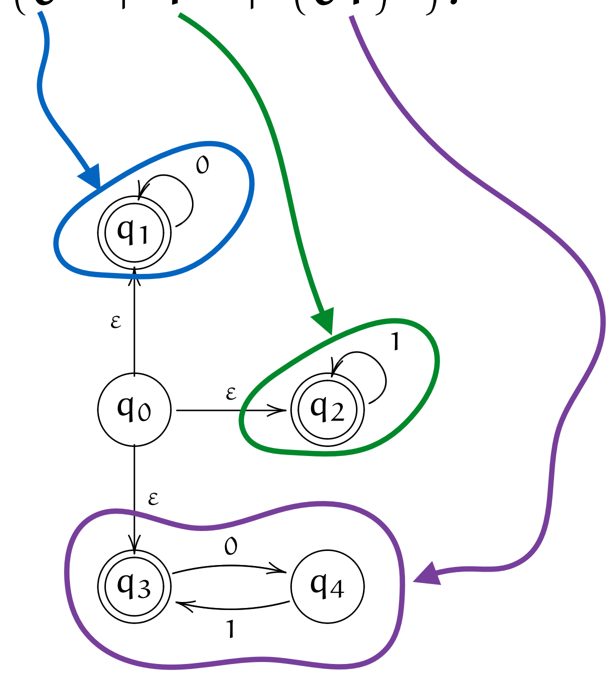


Esempio: from RE to ε-*NFA*

$$L = \{\ x \mid \exists n \in \mathbb{N}.\ x = 0^n \ \vee\ x = 1^n \ \vee\ x = (01)^n\ \}$$

$$(0^* + 1^* + (01)^*).$$

ε-*NFA*



What have we shown?

- Regular expressions, finite state automata and regular grammars are really different ways of expressing the same thing.
- In some cases you may find it easier to start with one and move to the other
  - E.g., the language of an even number of one's is typically easier to design as a NFA or DFA and then convert it to a RE

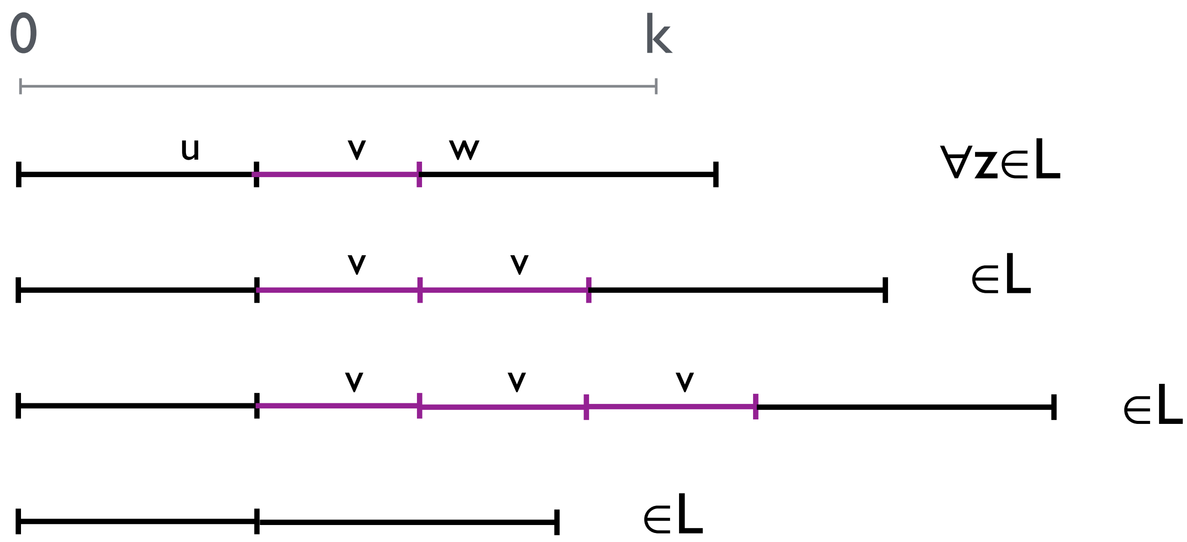
Not all languages are regular!

- $L=\{ a^n b^n \mid n \in Nat \}$

Pumping Lemma

Given L an infinite regular language then there exists an integer k such that for any string $z \in L. |z| \geq k$ it is possible to split z into 3 substrings

$$z = uvw \text{ with } |uv| \leq k, |v| > 0 \text{ such that } \forall i \in \mathbf{N}, uv^i w \in L$$



Negating the PL

The PL gives a necessary condition, that can be used to prove that a language is not regular!

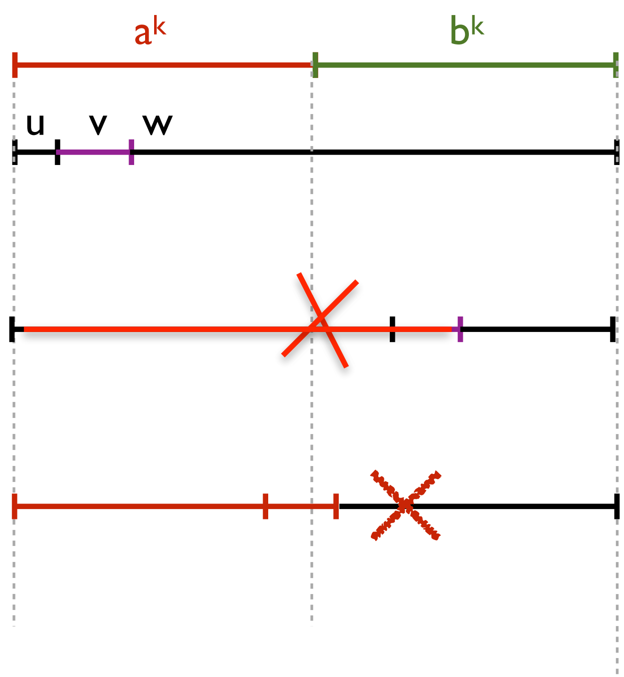
If $\forall k \in \mathbf{N} \;\; \exists z \in L. |z| \geq k$ for all possible splitting

$z = uvw$ with $|uv| \leq k, |v| > 0 \;\; \exists i \in \mathbf{N}$ such that $uv^i w \notin L$

then L is not a regular language!

Esempio

- L={ $a^n b^n$ | n ∈ Nat }, consider k ∈ N
- Let z = $a^k b^k$

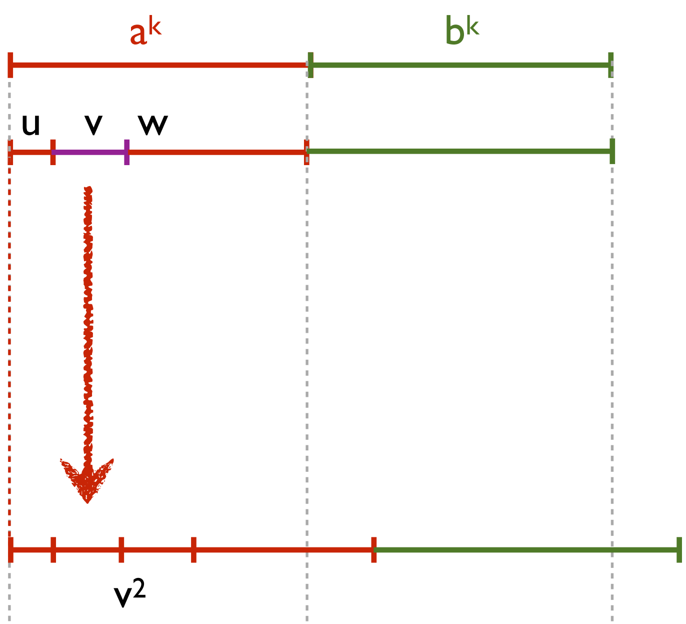


z∈L, |v|=i,

|uv| > k

Esempio

- $L=\{ a^n b^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k$



$z \in L$, $|v|=i$,

$a^{k+i} b^k \notin L$

$i \neq 0$

Property of Regular languages

The regular languages are closed with respect to the union, concatenation and Kleene closure.

The complement of a regular language is always regular.

- The regular language are closed under intersection

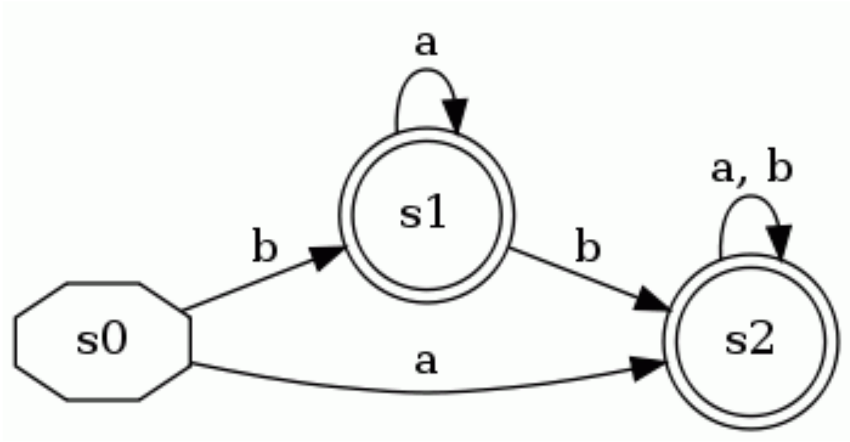
Decision Properties:

Approximately all the properties are decidable in case of finite automaton.

(i) Emptiness
(ii) Non-emptiness
(iii) Finiteness
(iv) Infiniteness
(v) Membership

DFA Minimization

- Some states can be redundant:
  - The following DFA accepts (a|b)+
  - State s1 is not necessary



DFA Minimization

- The task of DFA Minimization is to automatically transform a given DFA into a state-minimized DFA
  - Several algorithms and variants are known

DFA Minimization Algorithm

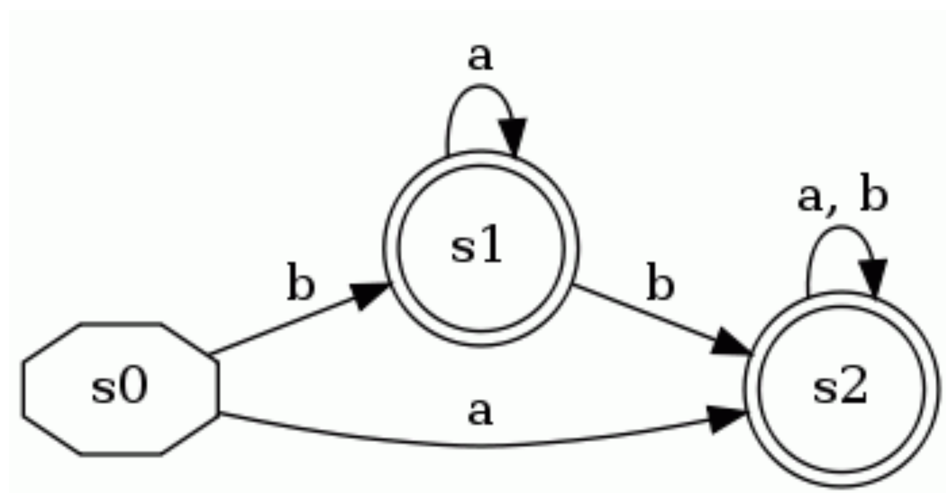
- Recall that a DFA $M=(Q, \Sigma, \delta, q_0, F)$
- Two states p and q are distinct if
  - p in F and q not in F or vice versa, or
  - for some a in Σ, $\delta(p, a)$ and $\delta(q, a)$ are distinct
- Using this inductive definition, we can calculate which states are distinct

DFA Minimization Algorithm

- Create lower-triangular table DISTINCT, initially blank
- For every pair of states (p,q):
  - If p is final and q is not, or vice versa
    - → DISTINCT(p,q) = ε
- Loop until no change for an iteration:
  - For every pair of states (p,q) and each symbol a
    - → If DISTINCT(p,q) is blank and DISTINCT(δ(p,a), δ(q,a)) is not blank
      - DISTINCT(p,q) = a
- Combine all states that are not distinct

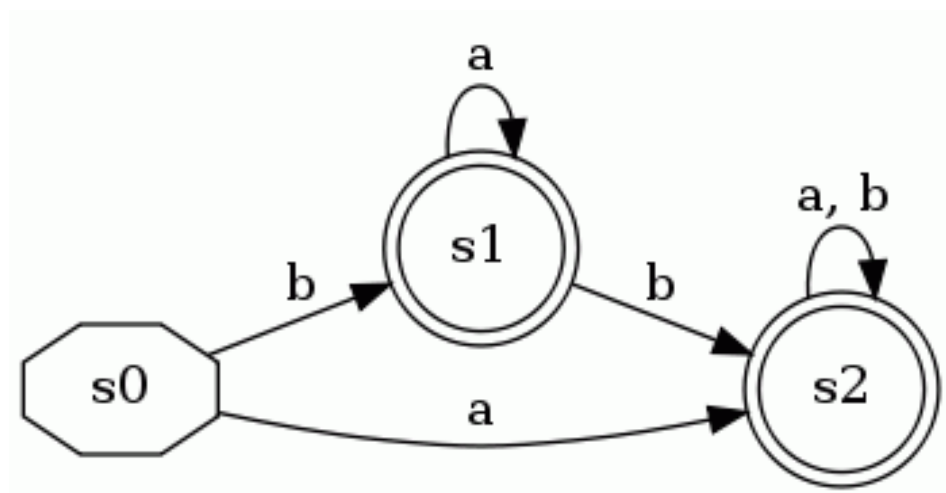
Very Simple Example

s0			
s1			
s2			
	s0	s1	s2



Very Simple Example

s0			
s1	ε		
s2	ε		
	s0	s1	s2



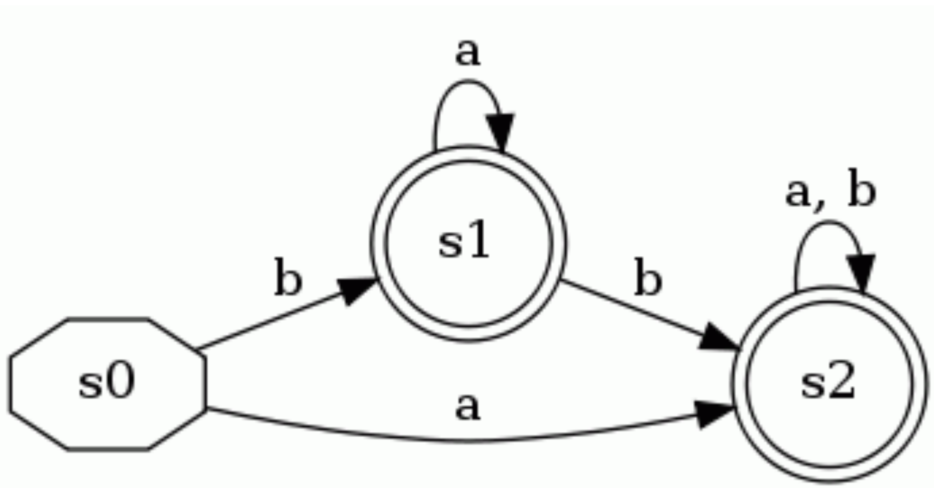
Label pairs with ε where one is a final state and the other is not

Very Simple Example

→ DISTINCT(p,q) is blank and DISTINCT($\delta(p,a)$, $\delta(q,a)$) is not blank

- DISTINCT(p,q) = a

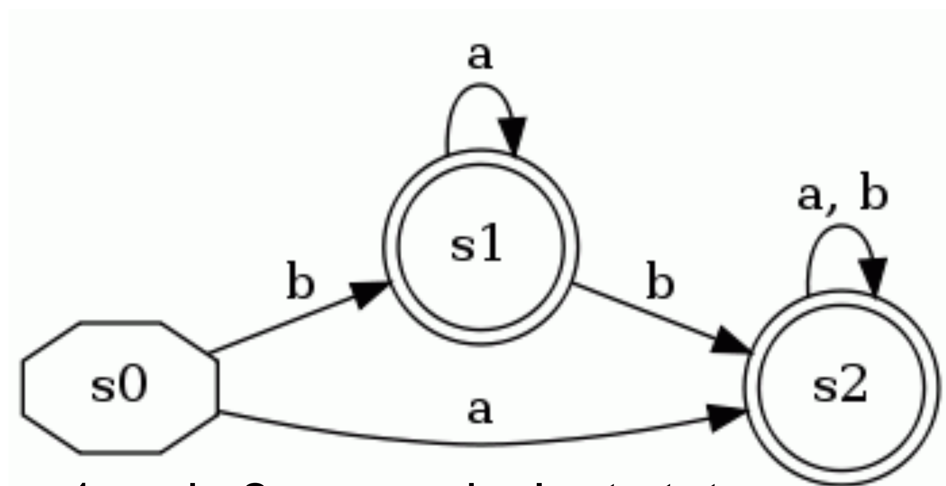
s0			
s1	ε		
s2	ε		
	s0	s1	s2



Main loop (no changes occur)

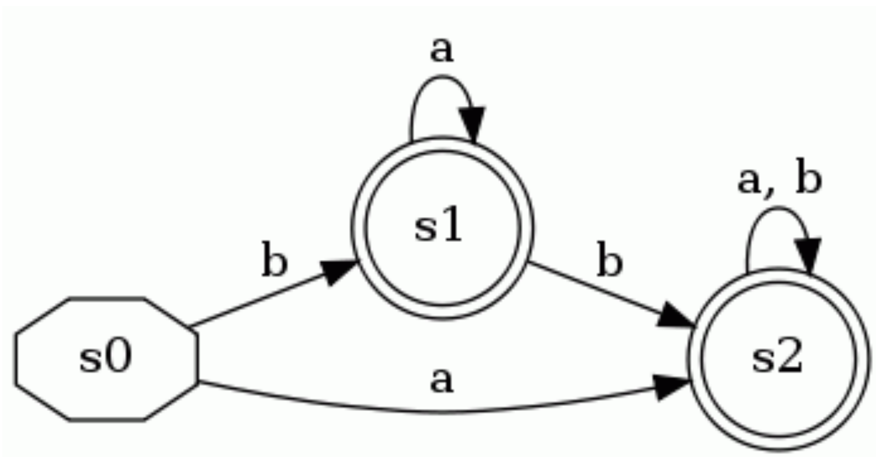
Very Simple Example

s0			
s1	ε		
s2	ε		
	s0	s1	s2

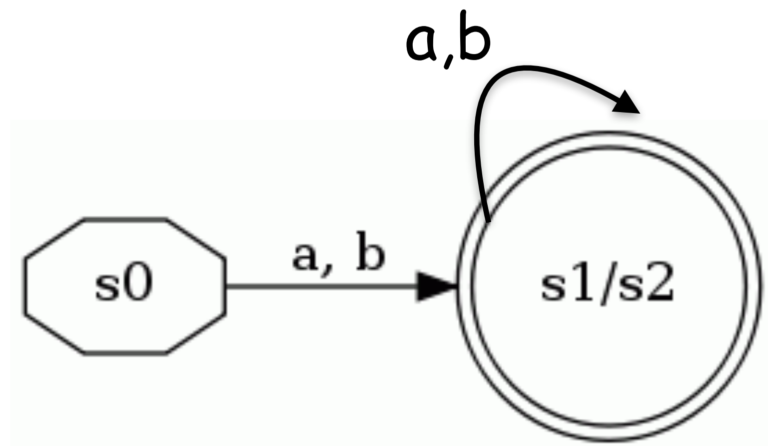


DISTINCT(s1, s2) is empty, so s1 and s2 are equivalent states

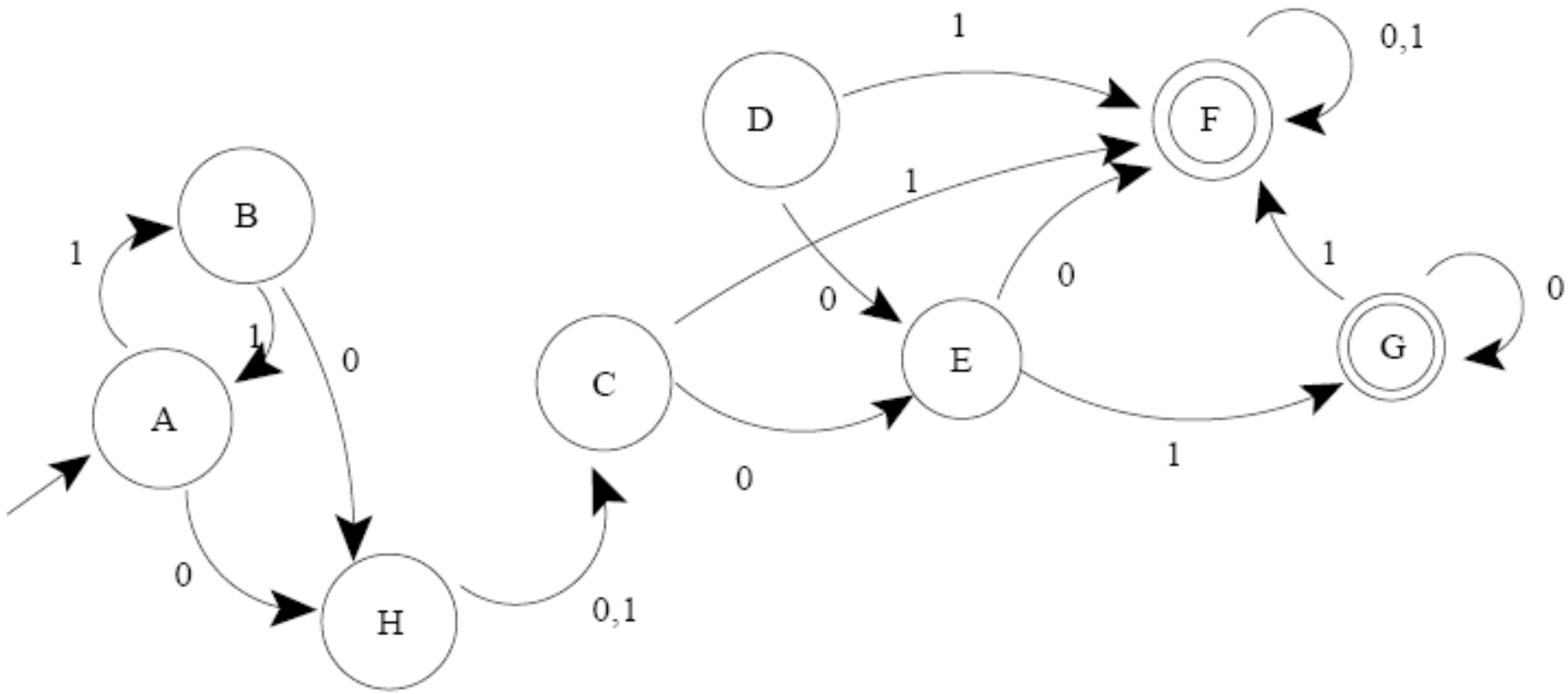
Very Simple Example



Merge s1 and s2

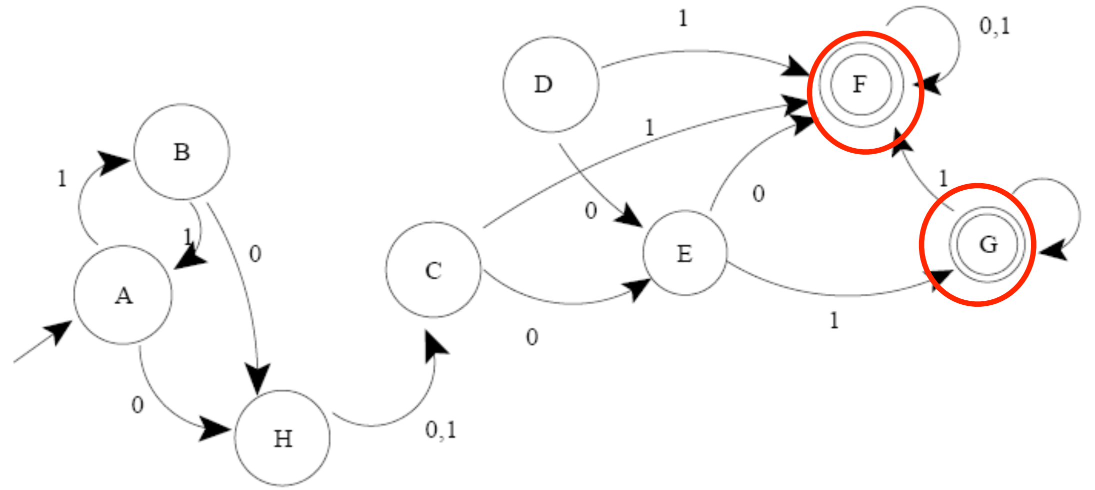


More Complex Example



More Complex Example

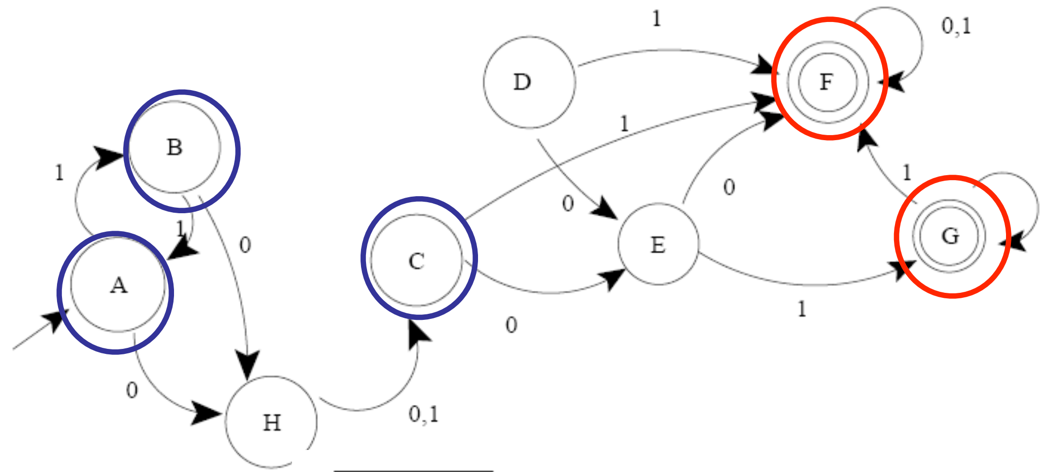
- Check for pairs with one state final and one not:



b							
c							
d							
e							
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h						ϵ	ϵ
	a	b	c	d	e	f	g

More Complex Example

- First iteration of main loop:

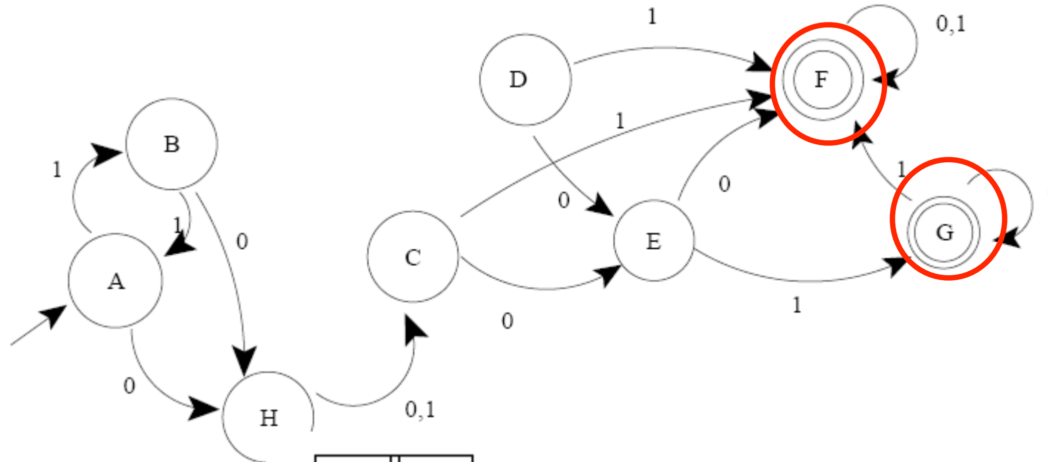


b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h			1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

b							
c							
d							
e							
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h						ϵ	ϵ
	a	b	c	d	e	f	g

More Complex Example

- Second iteration of main loop:



b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h	1	1	1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h			1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

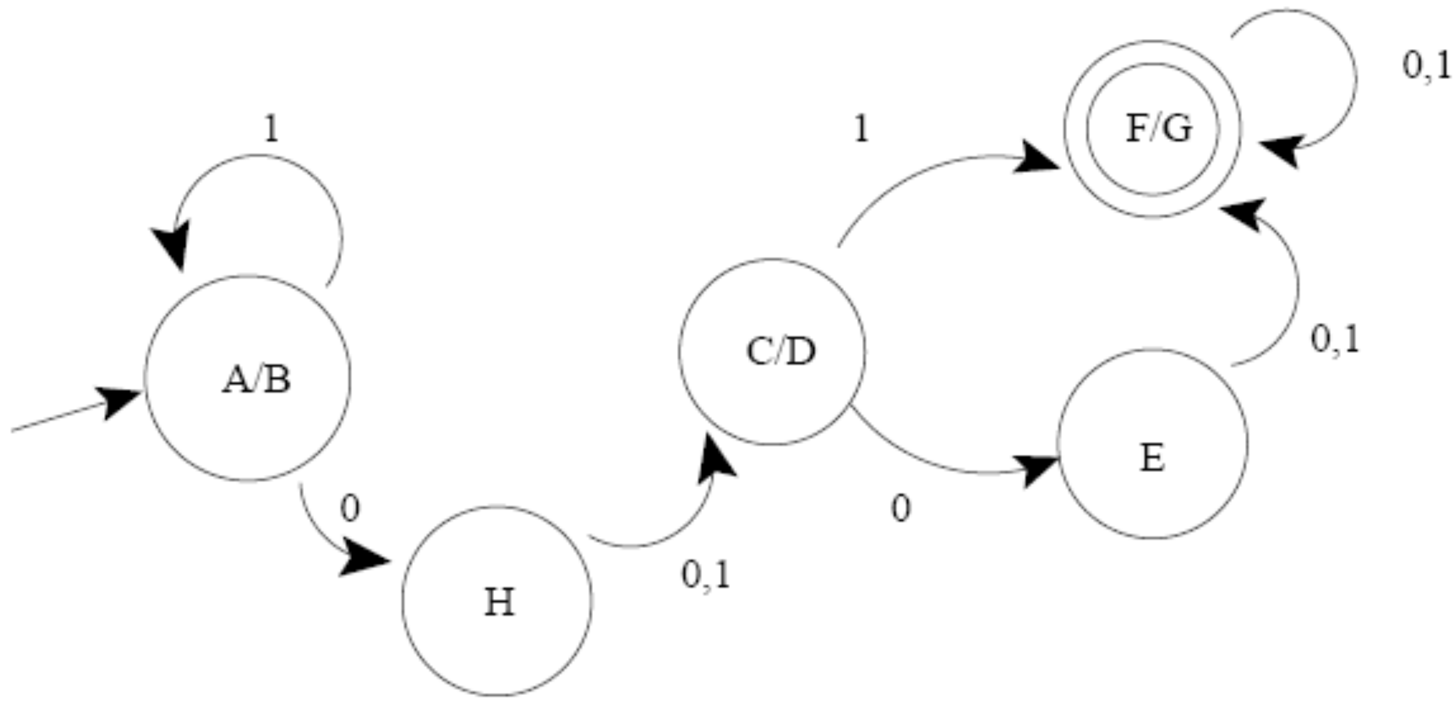
More Complex Example

- Third iteration makes no changes
  - Blank cells are equivalent pairs of states

b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h	1	1	1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

More Complex Example

- Combine equivalent states for minimized DFA:



Conclusion

- DFA Minimization is a fairly understandable process, and is useful in several areas
  - Regular expression matching implementation
  - Very similar algorithm is used for compiler optimization to eliminate duplicate computations
- The algorithm described is $O(kn^2)$
  - John Hopcraft describes another more complex algorithm that is $O(k\,(n \log n))$

Linguaggi Context Free

Context free Grammars

A Context free Grammar (Σ, N, S, P) is a generative grammar, where

- every production has the form $U \rightarrow V$

where U belongs to N and V belongs to $(\Sigma \cup N)^+$.

- only for the starting symbol S, we can have $S \rightarrow \varepsilon$

Example

$$G = \{\{E\}, \{or, and, not, (\,,\,), 0, 1\}, E, P\}$$

$$
\begin{aligned}
E &\mapsto 0 \\
E &\mapsto 1 \\
E &\mapsto (E \text{ or } E) \\
E &\mapsto (E \text{ and } E) \\
E &\mapsto (\text{not } E)
\end{aligned}
$$

Esempio

$$S \rightarrow 0S1 \mid \varepsilon$$

$$\{0^n 1^n : n \geq 0\}$$

Example

$$S \rightarrow \varepsilon | 0 | 1 | 0S0 | 1S1$$

$$z = \{x \in \{0,1\}^* \mid x = x^R\}$$

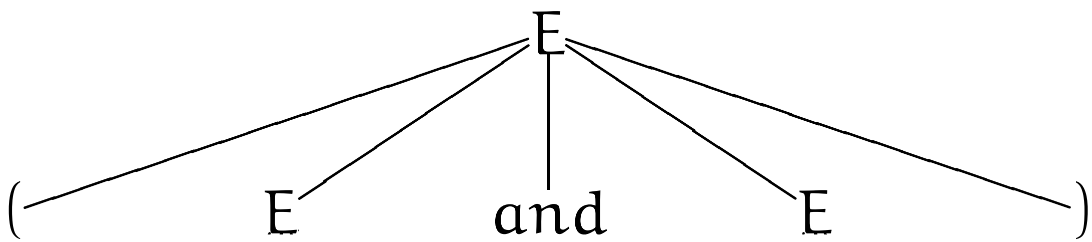
Parse tree

Given a grammar (Σ, N, S, P).
The parse tree is the graph representation of a derivation, which can be defined in the following way:

- every node has a label in Σ U N U {ε},
- the label of the root and of every internal node belongs to N,
- if a node is labeled with A and has m children labeled with X1,. …, Xk then the production A->X1…Xk belongs to P,
- if a node is labeled with ε then is a leaf and is an only child,
- Every leaf is labelled with a symbol in Σ U {ε}.

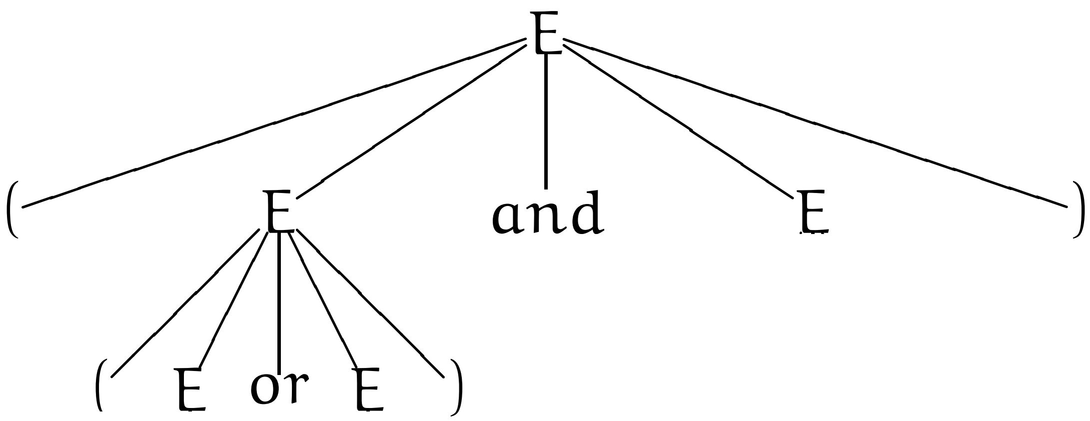
Example

$$E \mapsto 0|1|(E \text{ or } E)|(E \text{ and } E)|(\text{not } E)\,.$$



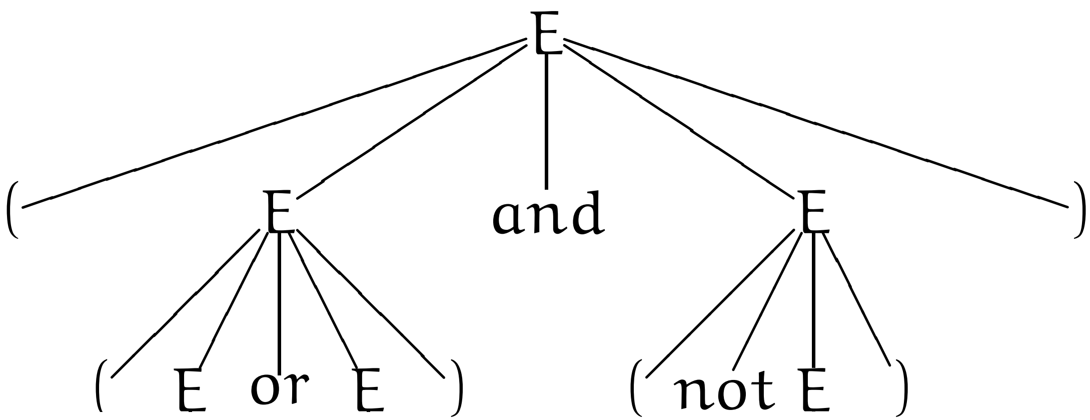
Example

$$E \mapsto 0 | 1 | (E \text{ or } E) | (E \text{ and } E) | (\text{not } E) \, .$$



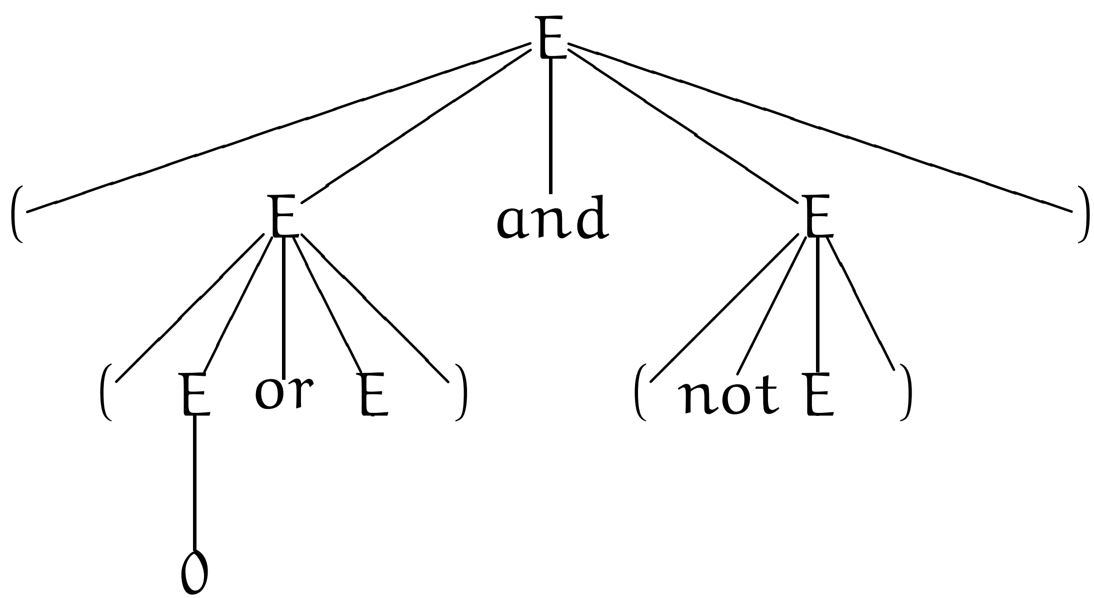
Example

$$E \mapsto 0|1|(E \text{ or } E)|(E \text{ and } E)|(\text{not } E).$$



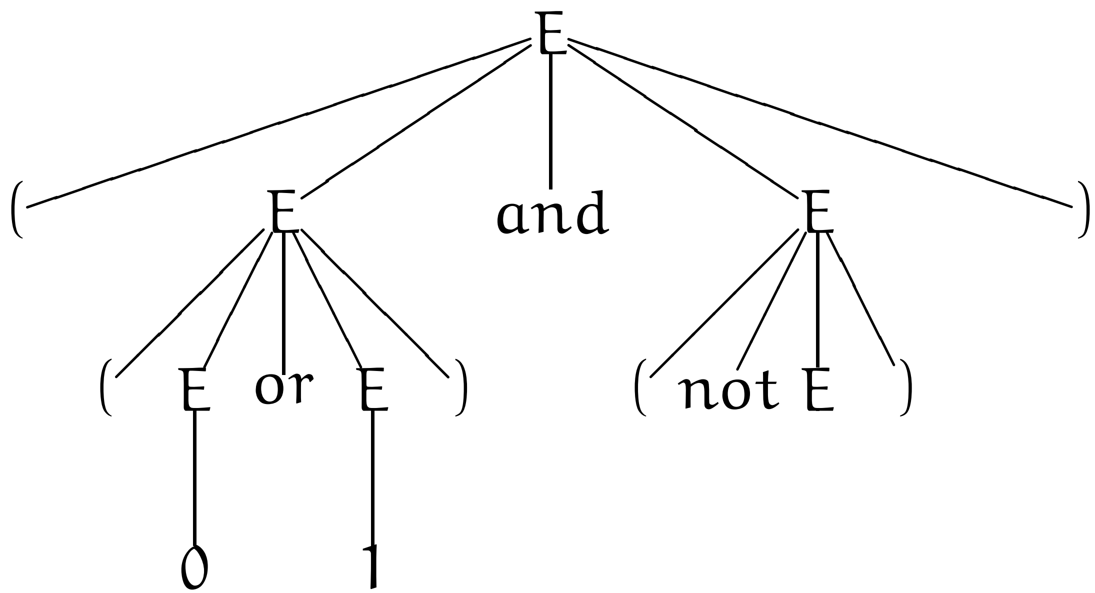
Example

$$E \mapsto 0 | 1 | (E \text{ or } E) | (E \text{ and } E) | (\text{not } E) \,.$$



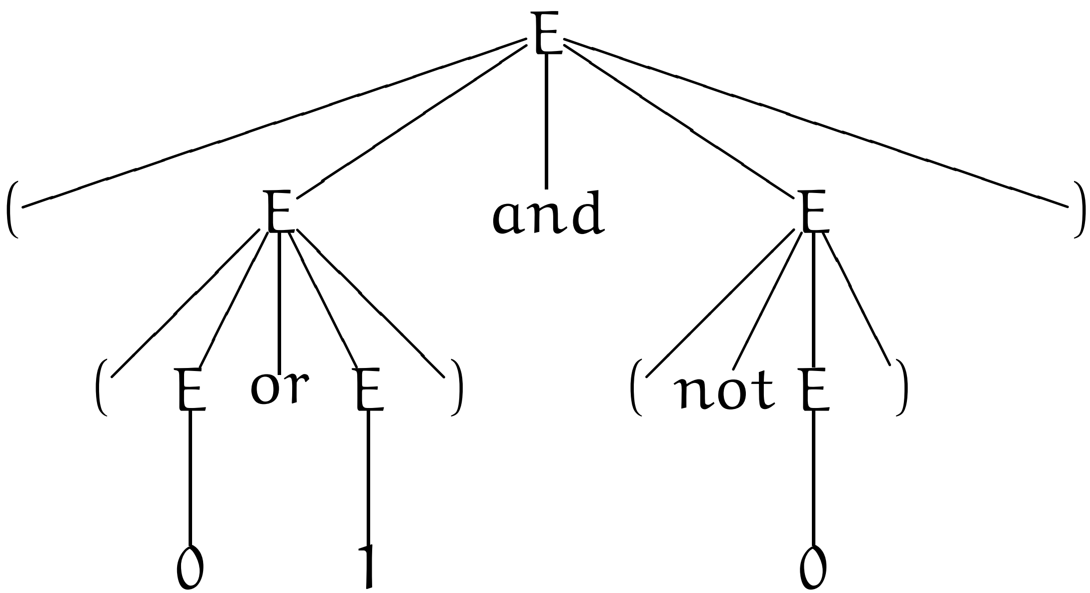
Example

$$E \mapsto 0 | 1 | (E \text{ or } E) | (E \text{ and } E) | (\text{not } E) \; .$$



Example

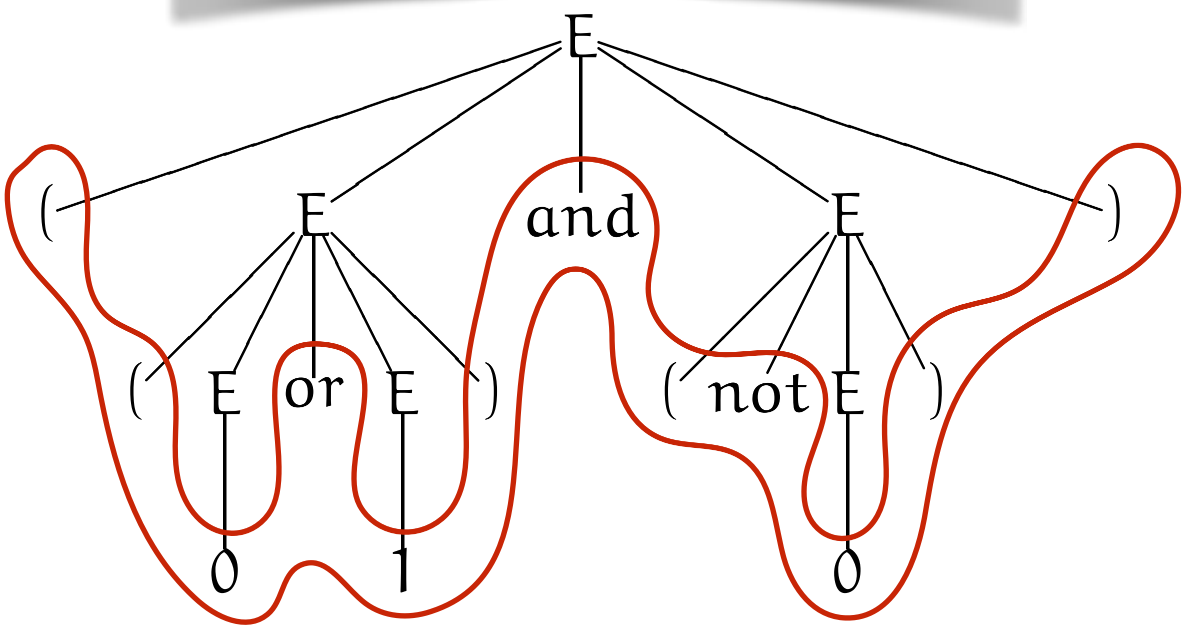
$$E \mapsto 0 | 1 | (E \text{ or } E) | (E \text{ and } E) | (\text{not } E) \,.$$



Example

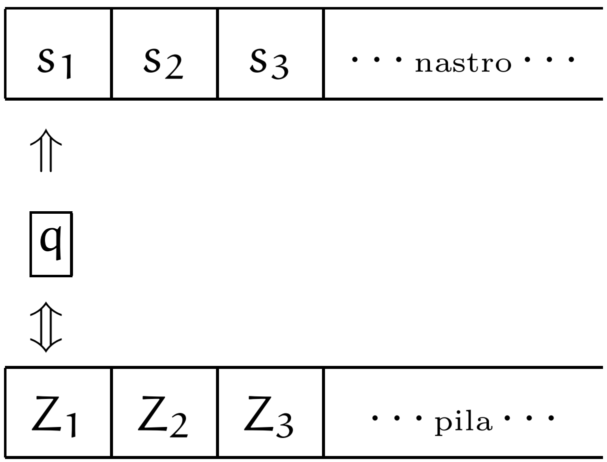
$$E \mapsto 0 | 1 | (E \text{ or } E) | (E \text{ and } E) | (\text{not } E) \,.$$

$$w = ((0 \text{ or } 1) \text{ and } (\text{not } 0))$$



Pushdown Automata I

The pushdown automaton are NDA with epsilon transitions and the stack



input stream

stack

Alphabet of stack symbols: R

Pushdown Automata II

The PDA can only access to the information in a first-in first-out way.

The stack head always scans the top symbol

It performs the following basic operations:

Push: add new symbols at the top of the stack
Pop: read and remove the top symbol
Empty: verify if the stack is empty

Formalization of Pushdown Automata

They can be represented by M = (Q, Σ, R, δ, q0,Z0, F) where

- R is the alphabet of stack symbols,

- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times R \rightarrow \wp_f(Q \times R^*)$ is the transition function

  $\delta(q, a, X)$ gives a finite set of pairs (p, γ)

  $\gamma = \epsilon$ is a Pop action

- Z0 belonging to R is the starting symbol on the stack

Instantaneous Description

The evolution of the PDA is described by triples (q, w, γ) where;

- $q \in Q$ is the current state of the control unit
- $w \in \Sigma^*$ is the unread part of the input string
- $y \in R^*$ is the current contents of the PDA stack

A move from one instantaneous description to another will be denoted by

(q0, aw, Zr) ⊢> (q1, w, γr) iff (q1,γ) belongs to δ(q0,a,Z)

The language accepted by a pushdown automaton

Two ways to define a language:

- with empty stack (in this case F is the empty set)
- with final states F

$$
\begin{aligned}
L_p(M) &= \{x \in \Sigma^* : (q_0, x, Z_0) \mapsto_M^* (q, \varepsilon, \varepsilon), q \in Q\} \\
L_F(M) &= \{x \in \Sigma^* : (q_0, x, Z_0) \mapsto_M^* (q, \varepsilon, \gamma), \gamma \in R^*, q \in F\}
\end{aligned}
$$

Esempio

We will recognise the string when the input and stack are empty!

$$L = \{\ xcx^R |\ x \in \{a,b\}^*\ \}, \Sigma = \{a,b,c\}$$

$\langle \{q_0, q_1\}, \{a, b, c\}, \{Z, A, B\}, \delta, q_0, Z, \emptyset \rangle$ *APND*

q_0	ε	a	b	c
Z		q_0, ZA	q_0, ZB	q_1, ε
A				
B				

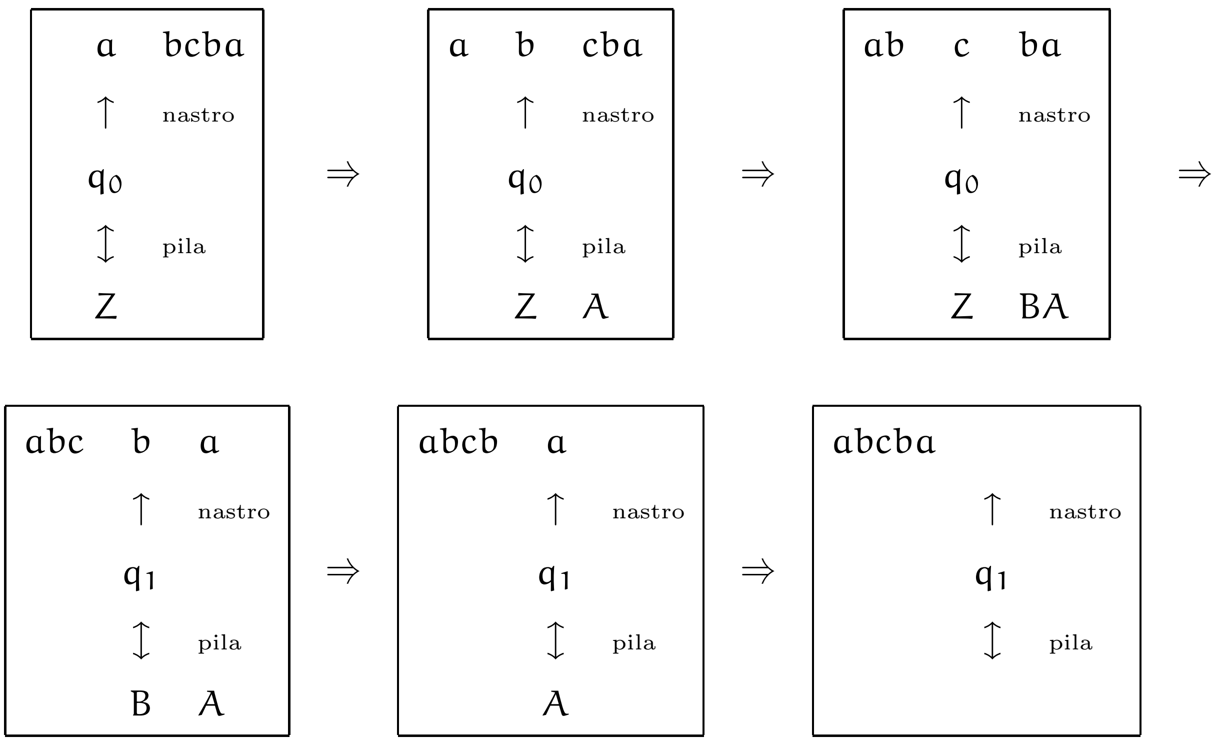
q_1	ε	a	b	c
Z		q_1, Z	q_1, Z	
A		q_1, ε	q_1, Z	q_1, Z
B		q_1, Z	q_1, ε	q_1, Z

Example: abcba

Remember:we will recognise the string when the input and stack are empty!

q_0	ε	a	b	c
Z		q_0, ZA	q_0, ZB	q_1, ε
A				
B				

q_1	ε	a	b	c
Z		q_1, Z	q_1, Z	
A		q_1, ε	q_1, Z	q_1, Z
B		q_1, Z	q_1, ε	q_1, Z



Example

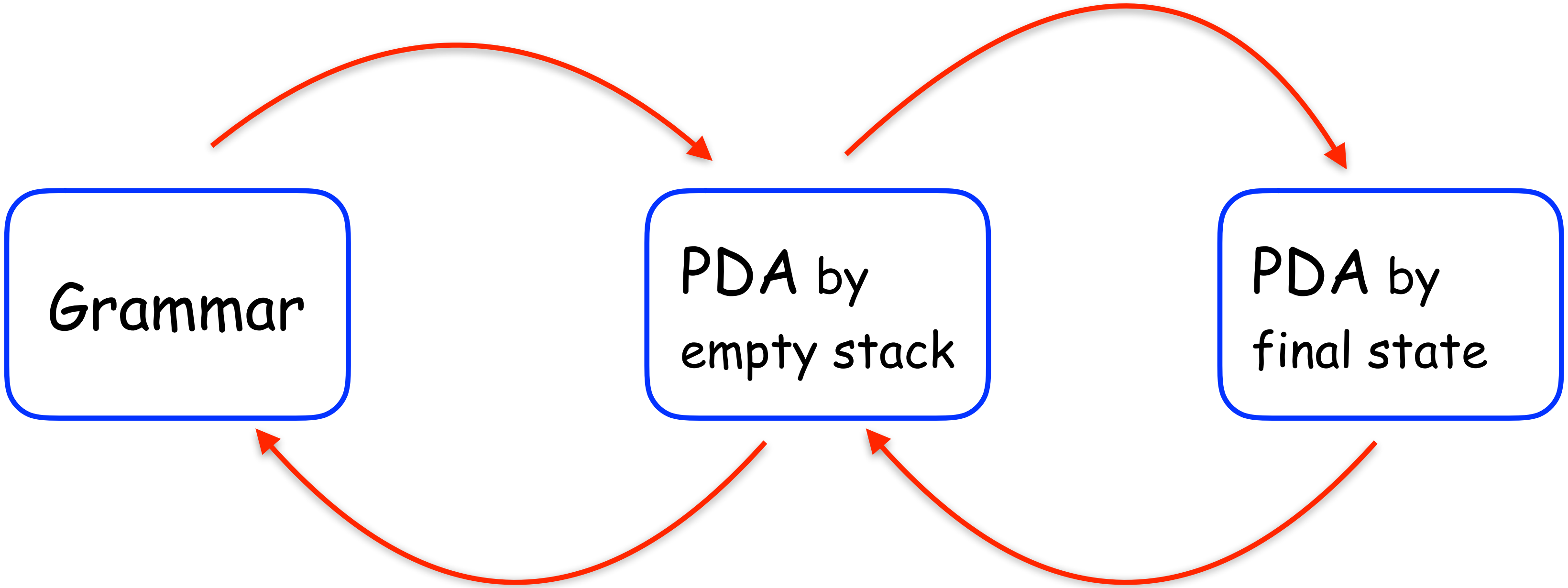
$$L = \{\ xx^R |\ x \in \{a,b\}^*\ \}, \Sigma = \{a,b\}$$

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $R = \{Z, A, B\}$

q_0	ε	a	b
Z	q_1, ε	q_0, AZ	q_0, BZ
A		q_0, AA q_1, ε	q_0, BA
B		q_0, AB	q_0, BB q_1, ε

q_1	ε	a	b
Z			
A		q_1, ε	
B			q_1, ε

Equivalence of PDA's and CGG's



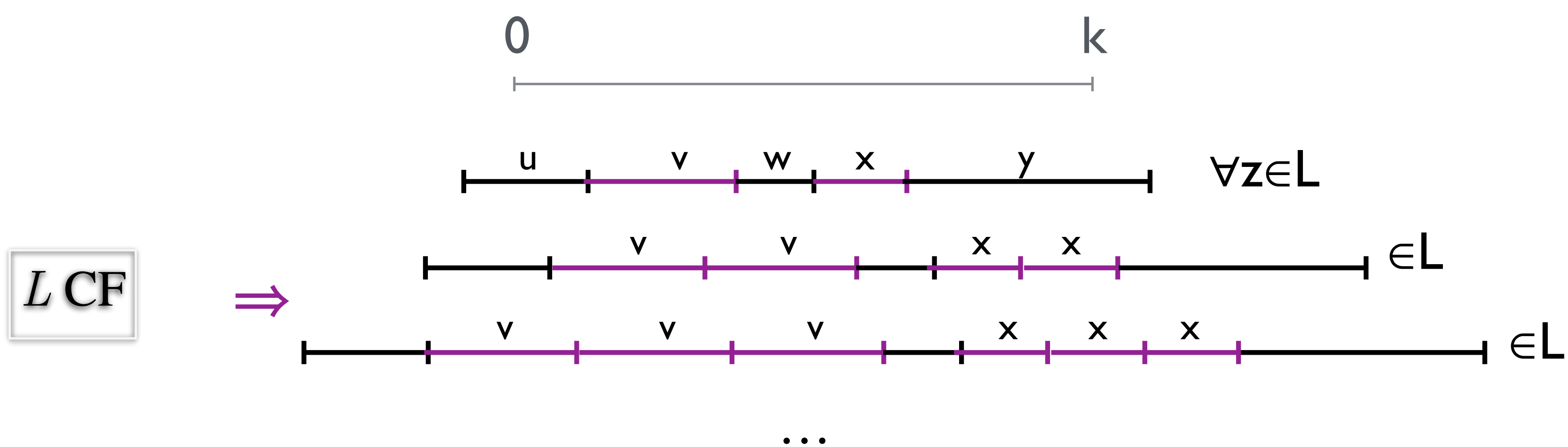
Unfortunately...

not all languages are Context Free !

Pumping Lemma for CF

Given a context free language L there exists an integer k such that for any string $z \in L. |z| \geq k$ it is possible to split z into 5 substrings

$$z = uvwxy \text{ with } |vwx| \leq k, |vx| > 0 \text{ such that } \forall i \in \mathbf{N}, uv^iwx^iy \in L$$



Negating the PL for CF

The PL for CF gives a necessary condition, that can be used to prove that a language is not context free!

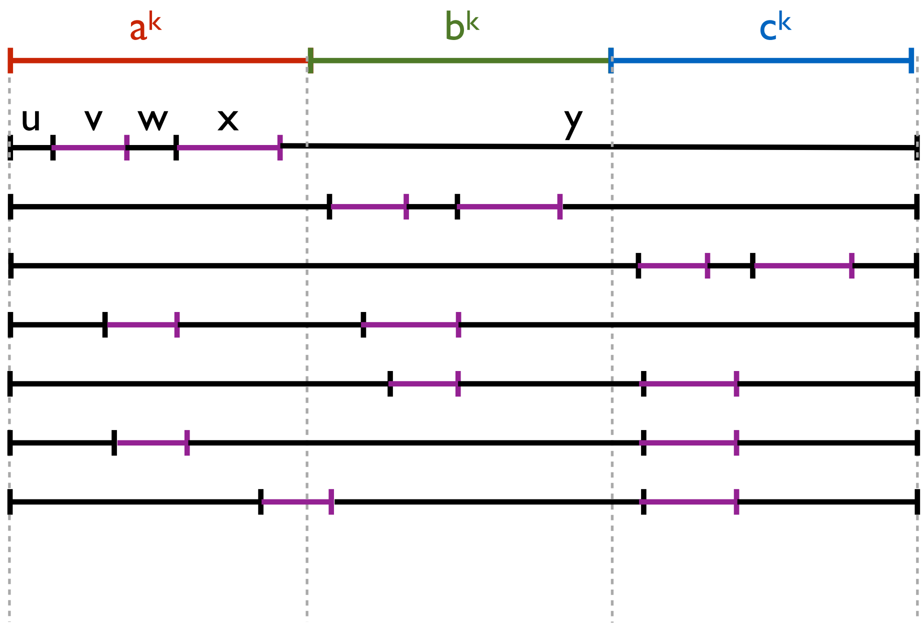
If $\forall k \in \mathbf{N} \ \ \exists z \in L. |z| \geq k$ for all possible splitting of the form

$$z = uvwxy \text{ with } |vwx| \leq k, |vx| > 0 \ \ \exists i \in \mathbf{N} \text{ such that } uv^iwx^iy \notin L$$

then L is not context free!

Example

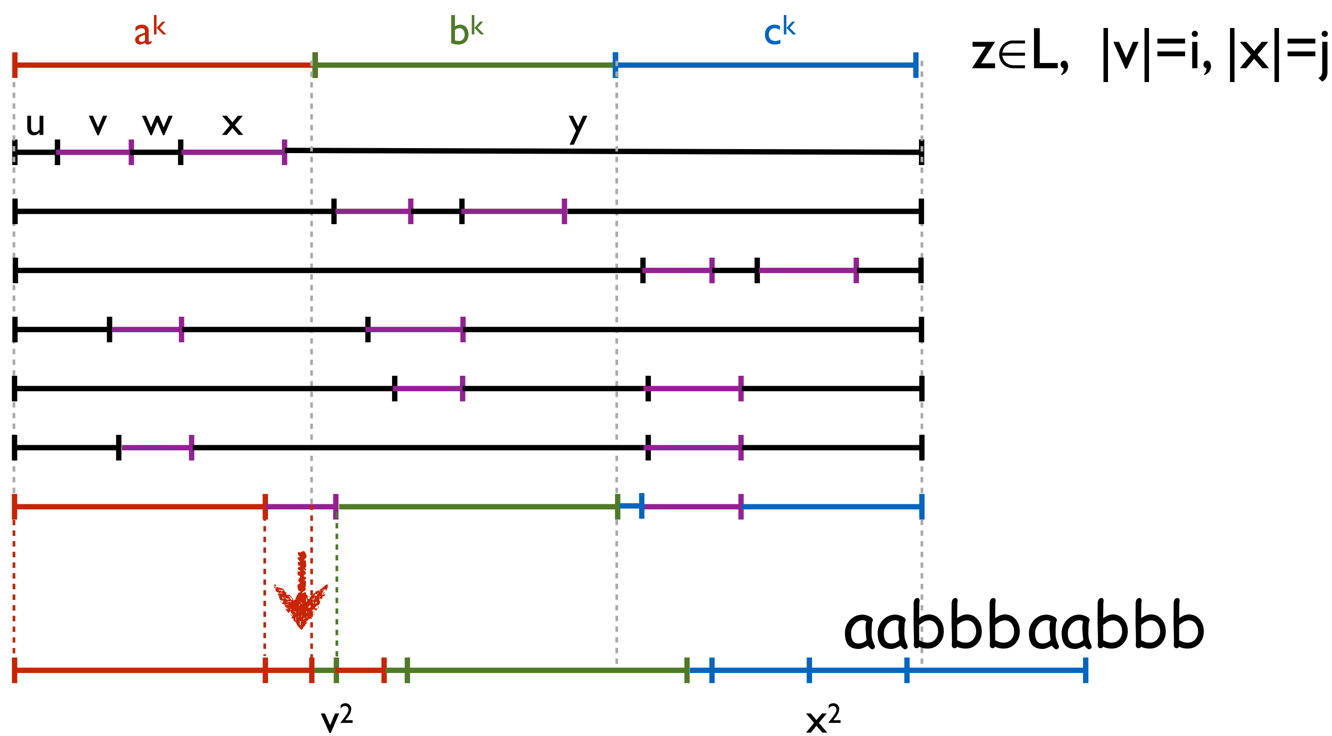
- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$



$z \in L$, $|v|=i$, $|x|=j$

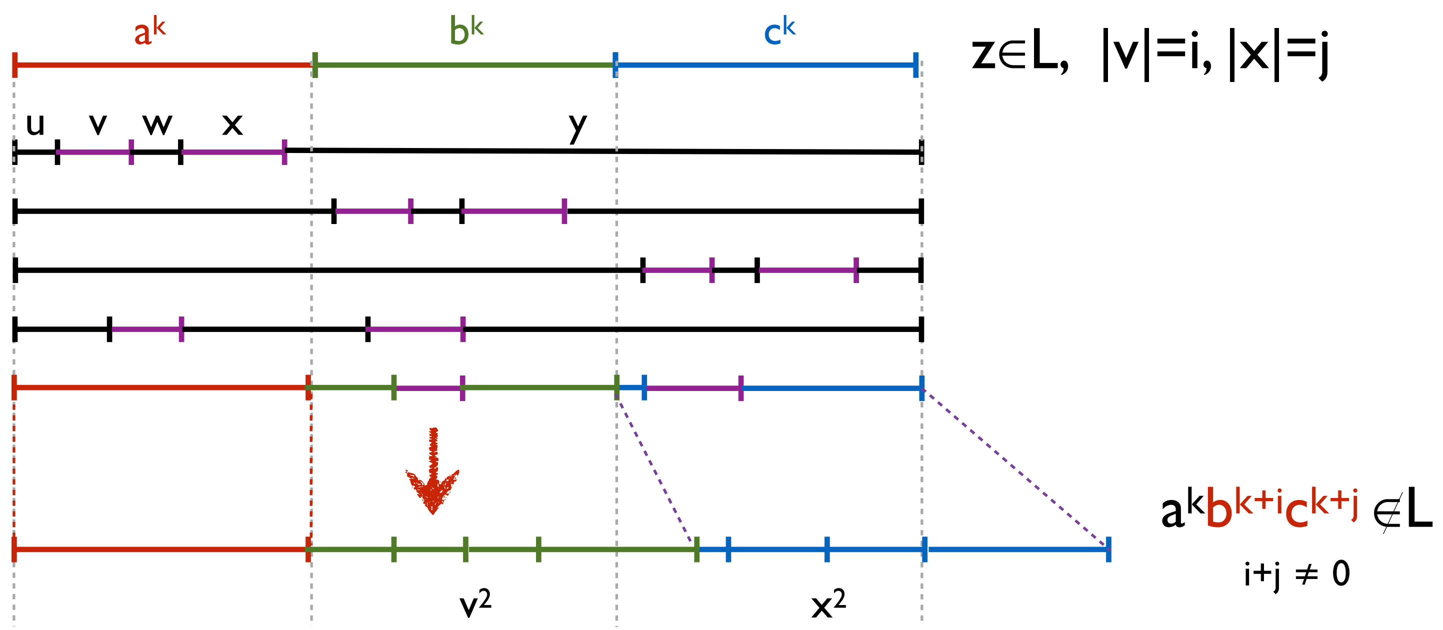
Example

- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$



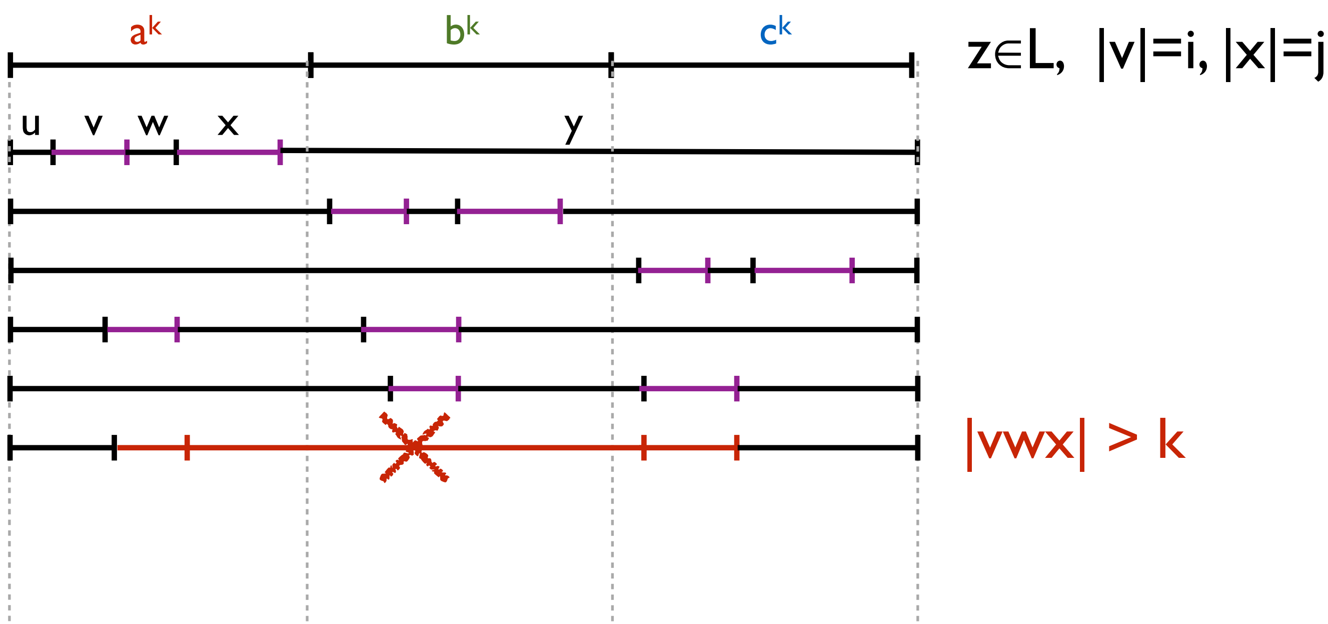
Example

- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$



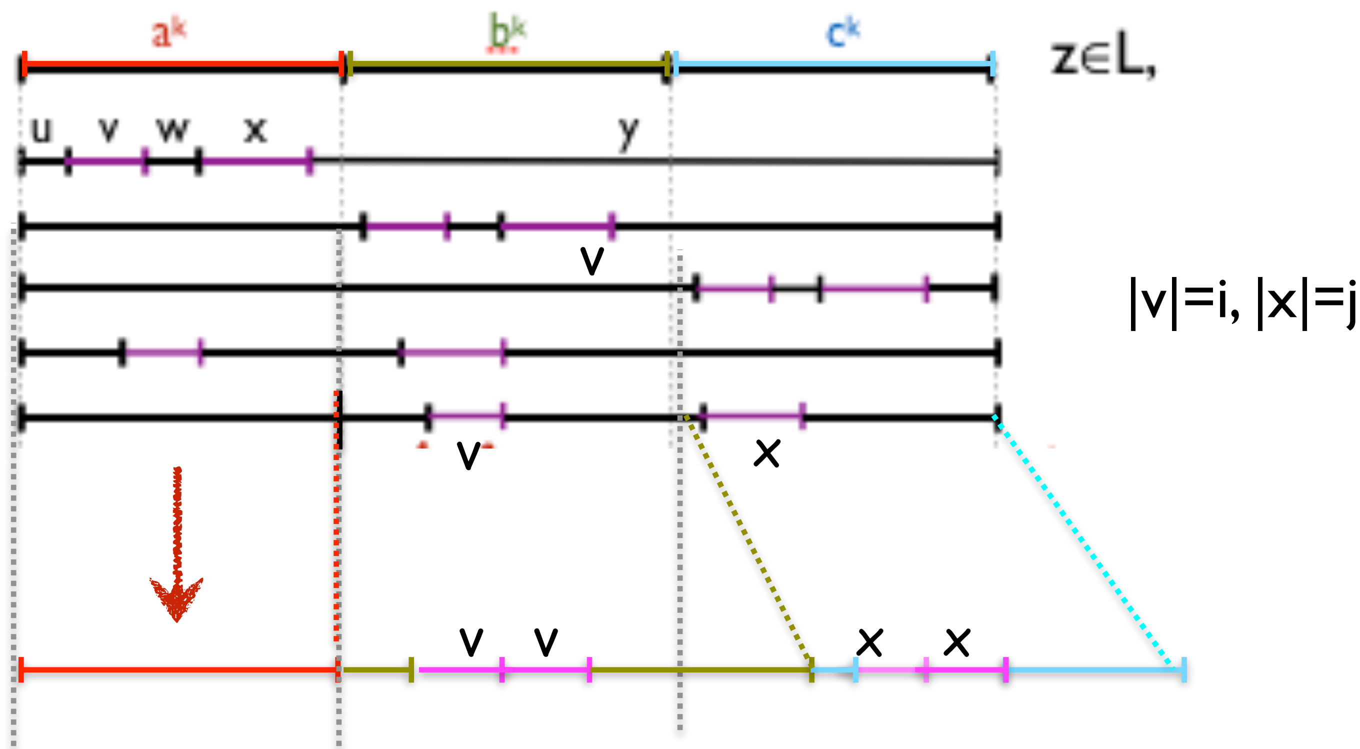
Example

- Let $L=\{ a^nb^nc^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^kb^kc^k$



Example

- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$

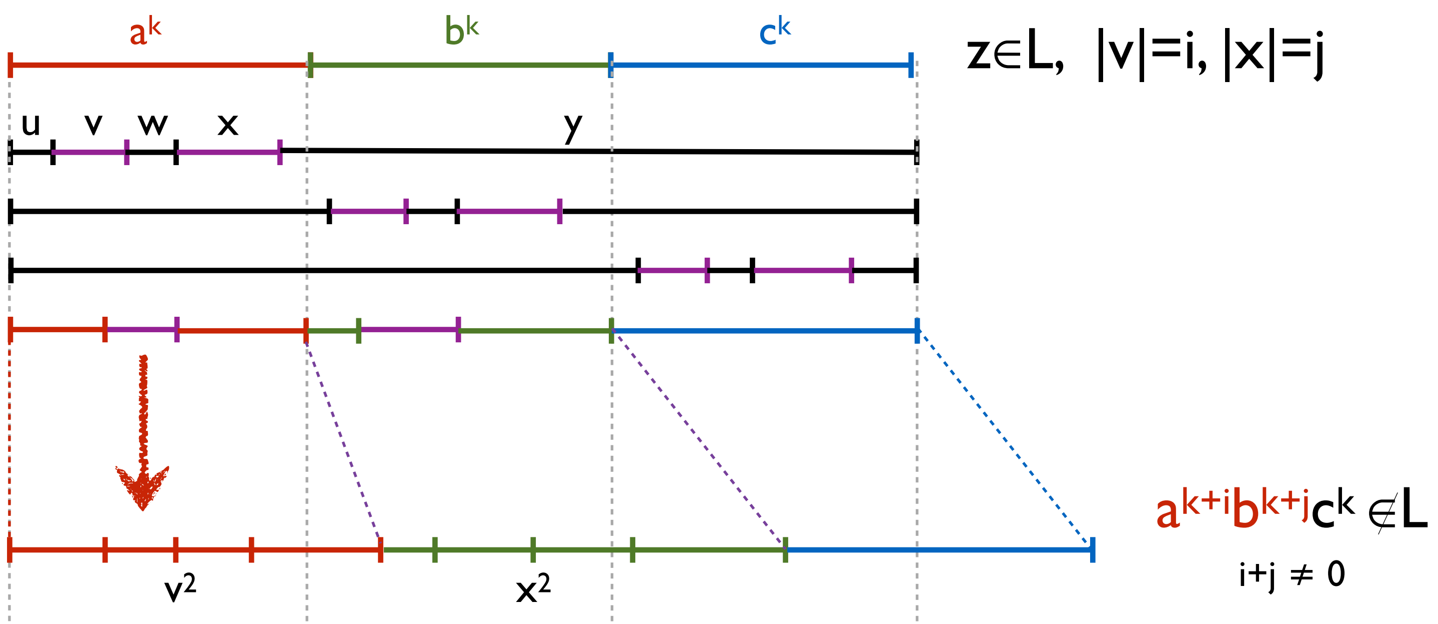


$a^k b^{k+i} c^{k+j} \notin L$

$i+j \neq 0$

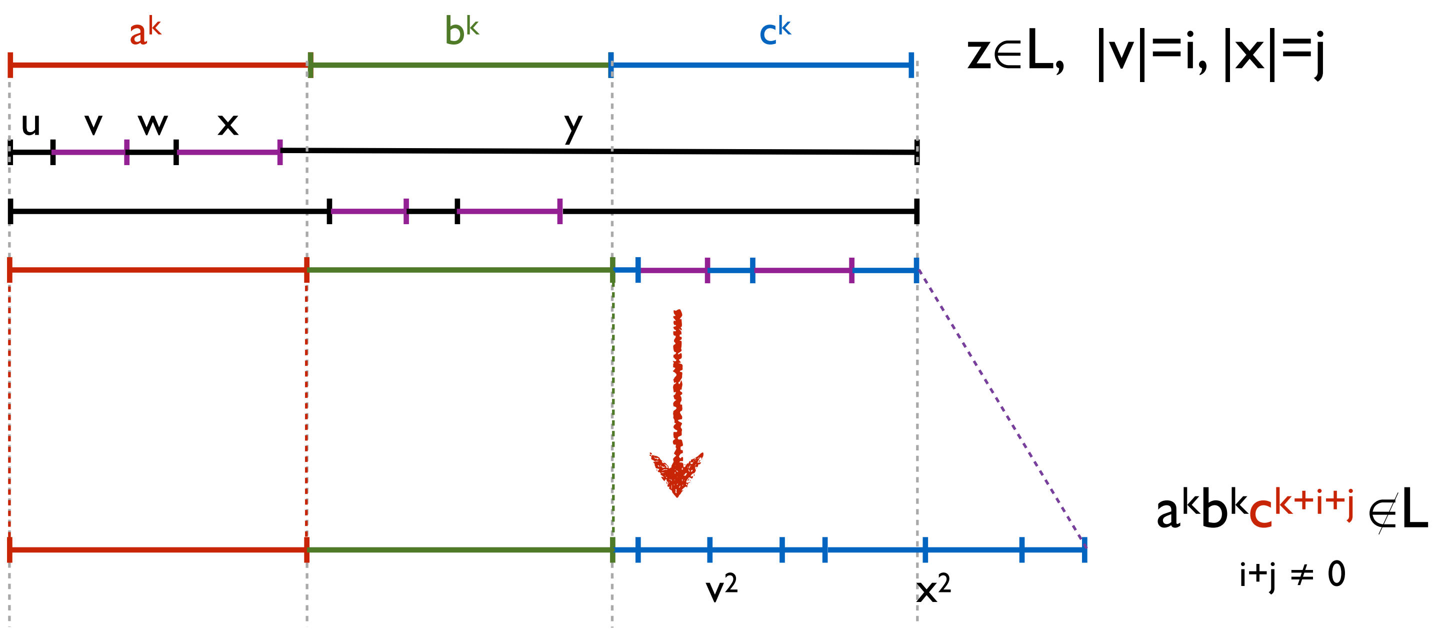
Example

- Let L={ $a^n b^n c^n$ | n ∈ Nat }, consider k ∈ N
- Let z = $a^k b^k c^k$



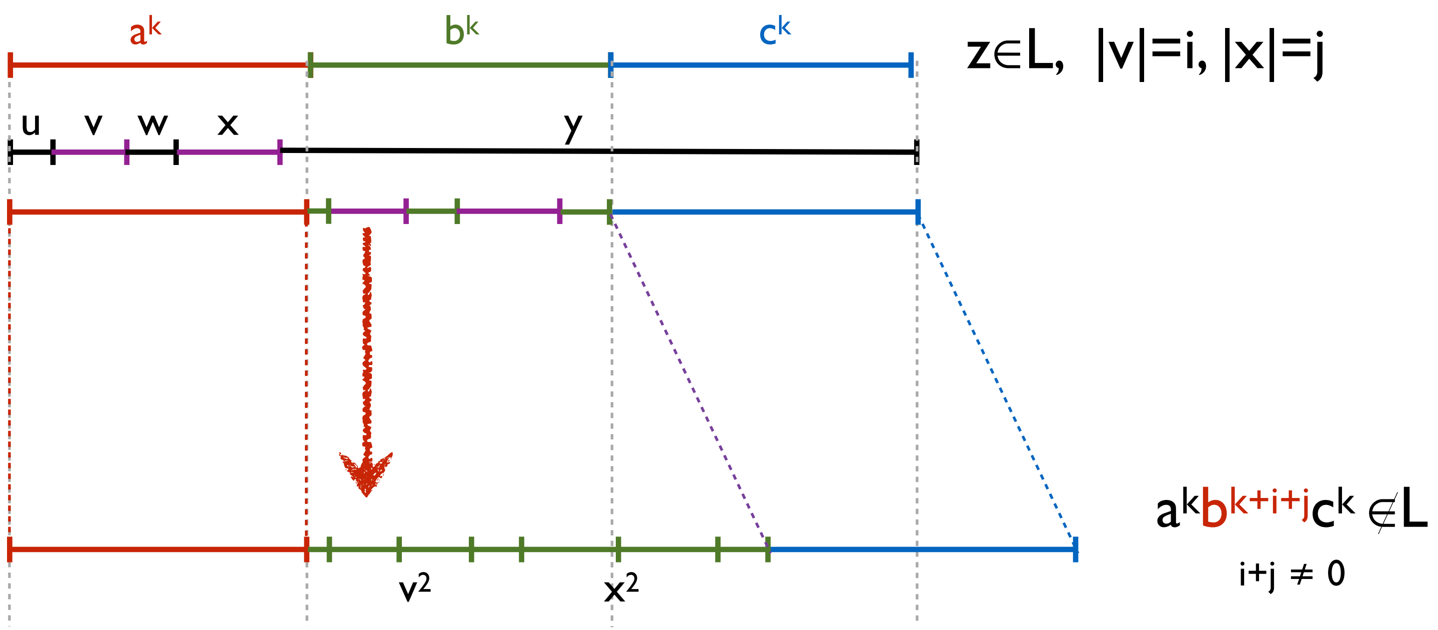
Example

- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$



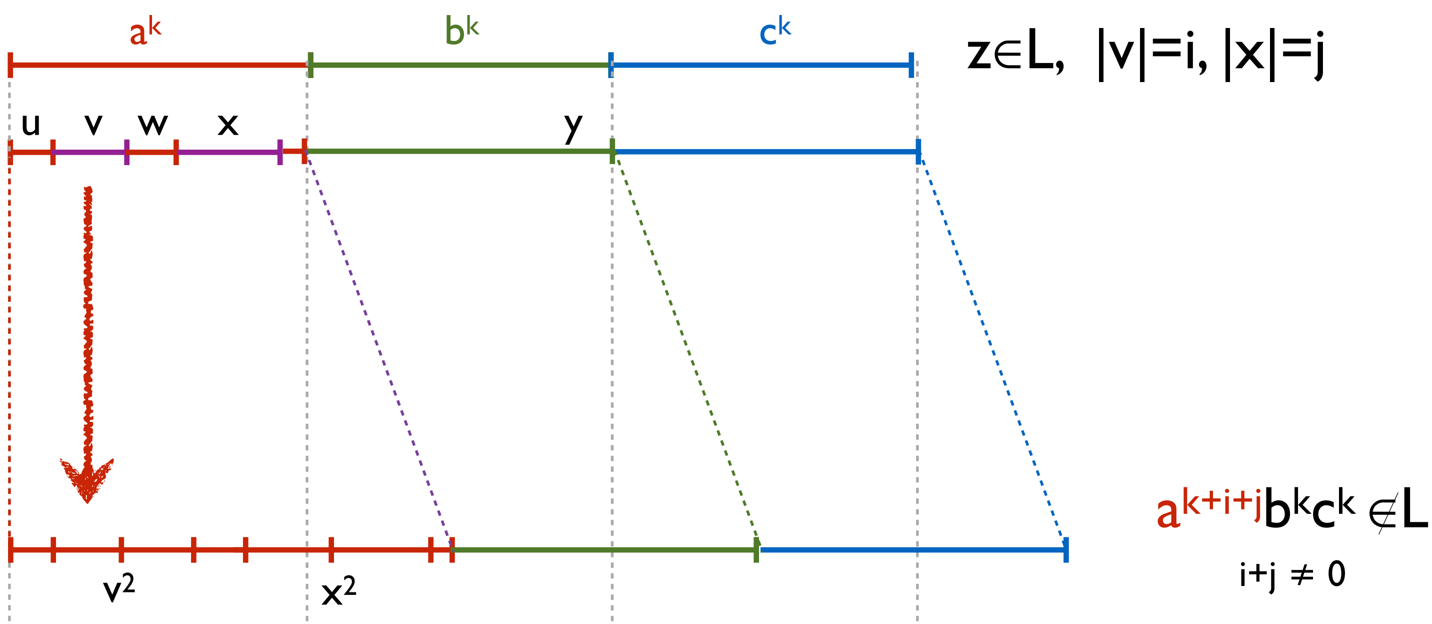
Example

- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$



Example

- Let $L=\{ a^n b^n c^n \mid n \in Nat \}$, consider $k \in N$
- Let $z = a^k b^k c^k$



Exercises: are these languages context free?

$$\{0^n 1^{3n} | n \geq 0\}$$

$$\{0^n 1^{kn} | n \geq 0 \text{ and } k \geq 0\}$$

$$\{w \in \{0,1\} | \text{ every prefix has more 0's than 1's}\}$$

$$\{a^i b^j c^k | i = j \text{ or } j = k\}$$

$$\{a^i b^j c^k | k \neq i + j\}$$

$$\{w \in \{a,b\}^* | w \neq vv\}$$

Properties of the CF languages

The CF languages are closed with respect to the union, concatenation and Kleene closure.

The complement of CF language is not always CF.

- The CF language are not closed under intersection

Decision Properties:

Approximately all the properties are decidable in case of CF

(i) Emptiness
(ii) Non-emptiness
(iii) Finiteness
(iv) Infiniteness
(v) Membership

Context Sensitive Grammar

Productions of the form U → V such that |U| <=|V|

Example

$$
\begin{aligned}
S &\to aSBC \mid aBC & \qquad bC &\to bc \\
CB &\to BC & \qquad cC &\to cc \\
bB &\to bb & \qquad aB &\to ab
\end{aligned}
$$

$$\{a^i b^i c^i : i \geq 1\}.$$

Complexity of Languages Problems

	Regular Grammar Type 3	Context Free Grammar Type 2	Context Sensitive Grammar Type 1	Unrestricted Grammar Type 0
Is W L(G)?	P	P	PSPACE	U
Is L(G) empty?	P	P	U	U
Is L(G1) L(G2)?	PSPACE	U	U	U

Examples of Language Hierarchy

The expressive power:

regular $\subset$ context-free $\subset$ context-sensitive $\subset$ phrase-structure

L1 = strings over {0, 1} with an even number of 1's is regular

L2 = $\{a^n b^n \mid n \geq 0\}$ is context-free, but not regular

L3 = $\{a^n b^n c^n \mid n \geq 0\}$ is context-sensitive, but not context-free

Relationships between Languages and Automata

A language is :

regular	iff accepted by	finite-state automata
context-free		pushdown automata
context-sensitive		linear-bounded automata
phrase-structure		Turing machine

Chomsky's Hierarchy