

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...

Strings

- An **alphabet** is a finite set of symbols

- Examples

$\Sigma_1 = \{a, b, c, d, \dots, z\}$: the set of letters in Italian

$\Sigma_2 = \{0, 1\}$: the set of binary digits

$\Sigma_3 = \{ (,) \}$: 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 $\Sigma_1 = \{a, b, c, d, \dots, z\}$

11011 is a string over $\Sigma_2 = \{0, 1\}$

))()((is a string over $\Sigma_3 = \{ (,) \}$

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

Operations on strings: length

- The **length** of a string x , denoted by $|x|$, is the number of symbols which compose x .

- Examples

$$|abfbz|=5$$

$$|110010|=6$$

$$|))(()|=7$$

$$|\varepsilon|=0$$

Operations on strings: concatenation and substrings

- 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 two 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 $y=\varepsilon$), s is called a **prefix** of x ;
when $x = ys$ (substring with $z=\varepsilon$), s is called a **suffix** of x ;
 ε is a prefix and a suffix of any string (including ε itself)

Power of an alphabet

- We define 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 $\Sigma = \{0,1\}$

$$\Sigma^0 = \{\epsilon\}$$

$$\Sigma^1 = \Sigma = \{0, 1\}$$

$$\Sigma^2 = \{00, 01, 11, 10\}$$

$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

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

$$\Sigma^+ = \{0, 1, 00, 01, 11, 10, 000, 001, 010, 011, 100, 101, 110, 111, \dots\}$$

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 represents a binary number divisible by 7

= {111, 10001, 10101, ...}

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

= { ϵ ,),))), ()),)()), ...}

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 and punctuation symbols

Operations on Languages

Union: $A \cup B$

Intersection: $A \cap B$

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

Complement: $\bar{A} = \Sigma^* - A$ where Σ^* is the set of all strings on Σ

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

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
 $\{\epsilon, 01, 0011, 000111, \dots\}$
- The set of strings with an equal number of 0's and 1's
 $\{\epsilon, 01, 10, 0011, 0101, 1001, \dots\}$
- The empty language \emptyset
- The language $\{\epsilon\}$ consisting of the empty string only

Remember $\emptyset \neq \{\epsilon\}$

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....

We can try to recognise when a word belongs to L

The generative approach: **Grammars**

Starting from a particular initial symbol, using the rewriting rules of the productions,
we **generate** the set of all the strings belonging to the language

Definition of Grammars

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

$$U \rightarrow V$$

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

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

A string $w \in \Sigma^*$ is generated by G if there exists a derivation starting from S and resulting in w obtained by rewriting the string using the productions in P

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

$$S \rightarrow \epsilon$$

$$S \rightarrow a$$

$$S \rightarrow aS$$

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

Example of a grammar

We want to describe L_1 the language of strings with an even number of 1's

L_1 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 L_1 iff it can be generated by the grammar

Grammar Example

Does the string 01010 belong to L1?

We need to find a derivation

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

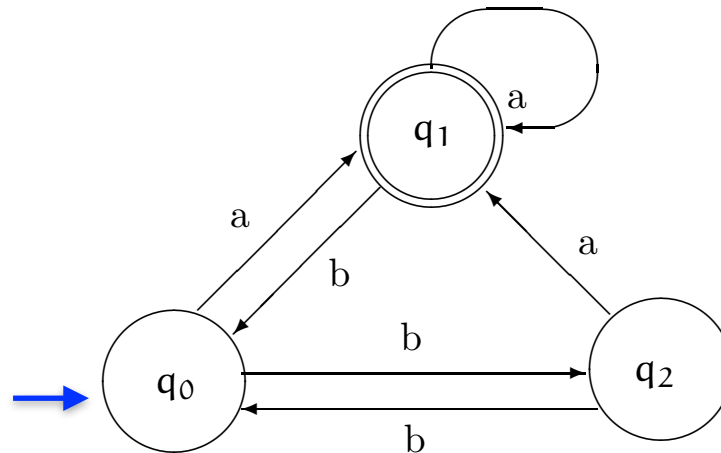
$$T \rightarrow 0T \mid 1S$$

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 to keep track of previous history.

input: baab



Back to our Problems

- Does the string w belong to the language L ?

We want to define a procedure to decide it!

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

It depends on the complexity of the language!!

Classification of 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 a type iff it admits a grammar of that type

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 $L(G_1) \equiv L(G_2)$?	PSPACE	U	U	U

P: decidable in polynomial time

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

U: undecidable

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 grammar, where

- every production has the form $A \rightarrow aB \mid a$
- only for the starting symbol S we can have $S \rightarrow \varepsilon$

Example

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

$S \rightarrow aS \mid aB$

$B \rightarrow bB \mid b$

$aaabb \in L(G)??$

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

S

Deterministic Finite Automata

A deterministic finite automaton (DFA) $(Q, \Sigma, \delta, q_0, F)$

Q a finite set of states

Σ a finite set Σ of symbols

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

q_0 the starting state

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

Deterministic Finite Automata

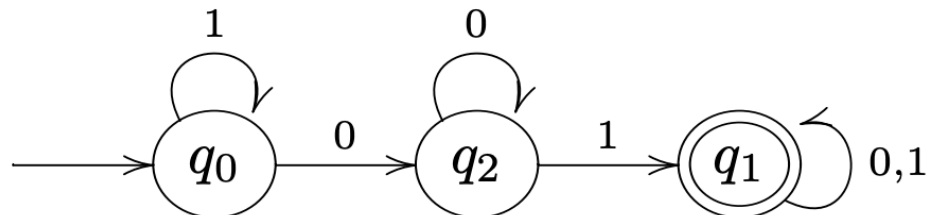
How to represent a DFA? With a **transition table**

	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

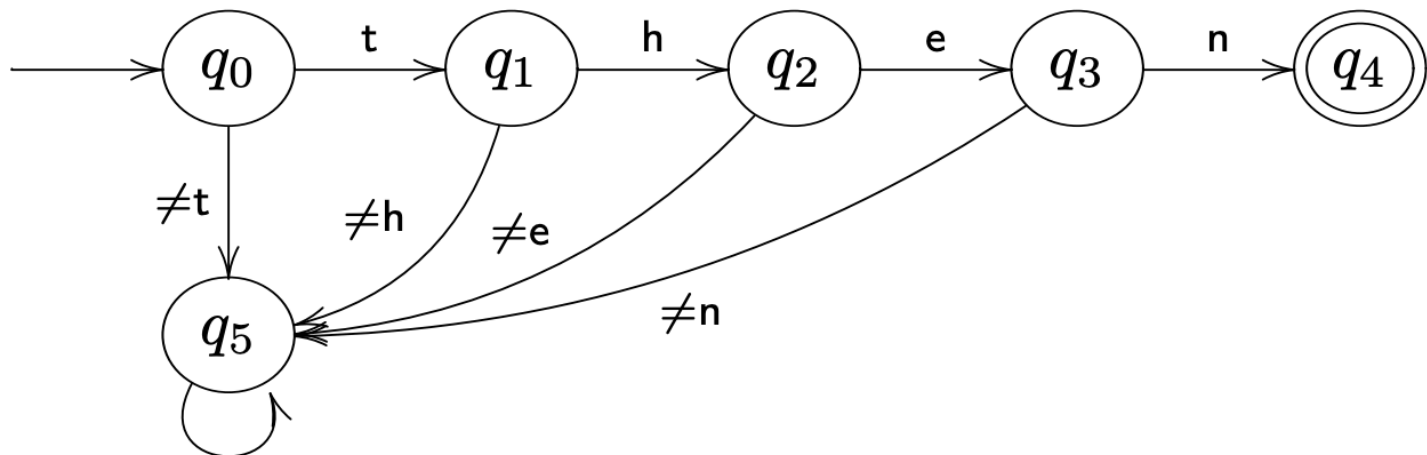
This defines the following transition system



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

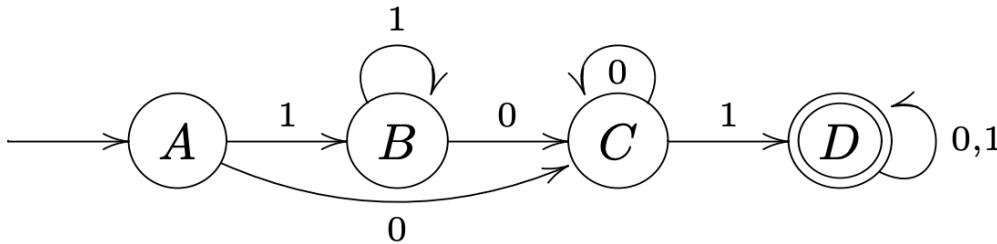
How DFA processes Strings

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
→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 $M = (Q, \Sigma, \delta, q_0, F)$ iff $\hat{\delta}(q_0, x) \in F$

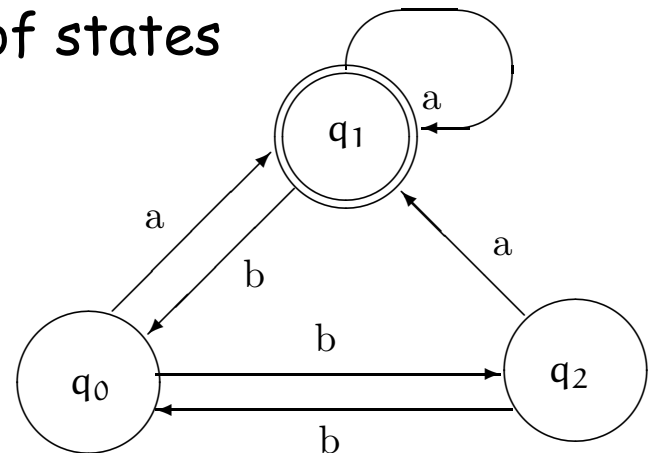
$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$$

Nondeterministic Finite Automata

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

Formally, a NFA is defined as $(Q, \Sigma, \delta, q_0, F)$ where the only difference is the transition function

$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ 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 $M=(Q, \Sigma, \delta, q_0, F)$ iff $\hat{\delta}(q_0, x) \cap F \neq \emptyset$

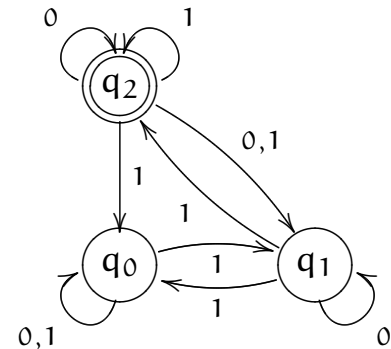
$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

- 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 }

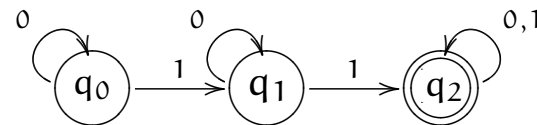
F = {q₂}.



NFA

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

	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

Different characterisation of Regular Languages

DFA

NFA

RG

RE

ϵ -NFA

- We formally will show how to pass from one characterization to another one

Roadmap: equivalence between NFA and RG

DFA

NFA \longleftrightarrow RG

RE

ϵ -NFA

From Regular Grammars to NFA

Theorem 1.

For each right grammar RG there is a non deterministic finite automaton NFA such that $L(RG)=L(NFA)$.

Construction Algorithm

Given a $RG=(\Sigma, N, S, P)$ construct a $NFA=(N \cup \{F\}, \Sigma, \delta, S, F')$ where F is a newly added state and if $F' = \{F\} \cup \{S\}$ if $S \rightarrow \epsilon$ belongs to P , $F' = \{F\}$, otherwise.

The transition function δ is defined by the following rules

- 1) For any $A \rightarrow a$ belonging to P , with a in Σ , set $\delta(A, a) = F$
- 2) For any $A \rightarrow aB$ 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:

$S \rightarrow aS \mid aB$

$B \rightarrow bB \mid b$

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

From NFA to Regular Grammars

Theorem 2

For each nondeterministic automaton NFA, there is one right grammar RG such that $L(RG)=L(NFA)$.

Construction Algorithm

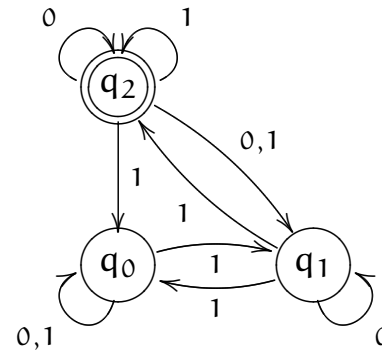
Given an automaton NFA = $(Q, \Sigma, \delta, q_0, F)$, construct a grammar $RG = (\Sigma, Q, q_0', P)$ according to the following steps:

- 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;
- 3) 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 }

F = {q₂}.



NFA

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

Exercises

Write the NFA for the following languages

- Strings over the alphabet $\{a,b,c\}$ containing at least one a and at least one b
- Strings of 0's and 1's whose tenth symbol from the right is 1
- The set of strings of 0's and 1's with at most one pair of consecutive 1's

and derive the corresponding grammars







Roadmap: equivalence between DFA and NFA



RE

ϵ -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 $N =$

we will build a $(Q_N, \Sigma, \delta_N, q_0, F_N)$

DFA $D =$

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

$$L(D) = L(N)$$

From NFA to a DFA

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

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

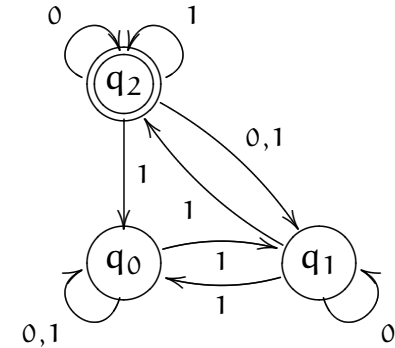
$$\forall P \in \mathcal{P}(Q_N) : \quad \delta_D(P, a) = \bigcup_{p \in P} \delta_N(p, a)$$

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

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

{q₀} {q₁} {q₂}

{q₀, q₁} {q₀, q₂} {q₁, q₂}

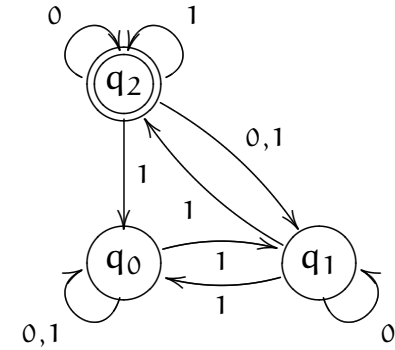
{q₀, q₁, q₂}

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

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

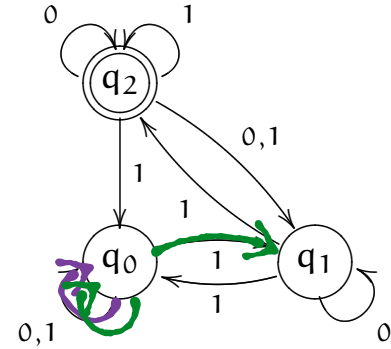
$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

Example

NFA

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



$\mathcal{P}(Q_N)$

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

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

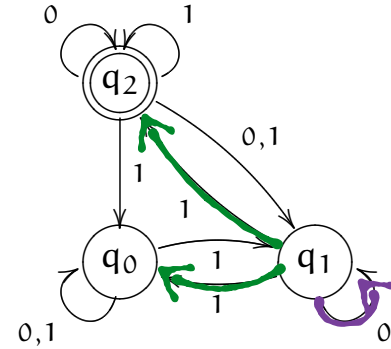
$\{q_0, q_1\}$ $\boxed{\{q_0, q_2\}}$ $\boxed{\{q_1, q_2\}}$

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

Example

NFA

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



$\mathcal{P}(Q_N)$

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

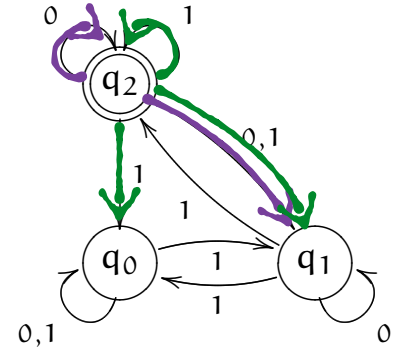
$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

Example

NFA

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



$\mathcal{P}(Q_N)$

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

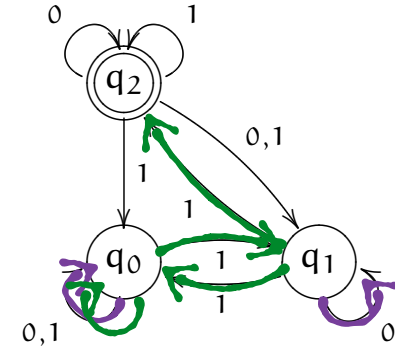
$\{q_0, q_1\}$ $\{q_0, q_2\}$ $\{q_1, q_2\}$

$\{q_0, q_1, q_2\}$

Example

NFA

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



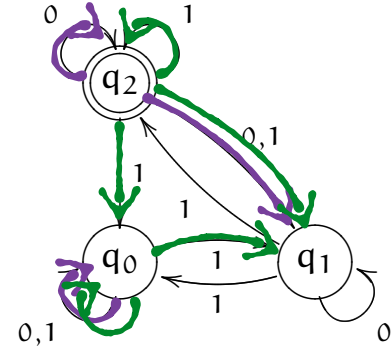
	\emptyset	0	1
\emptyset	\emptyset	\emptyset	\emptyset
q'_0	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q'_1	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
* q'_2	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q'_3	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$

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

Example

NFA

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



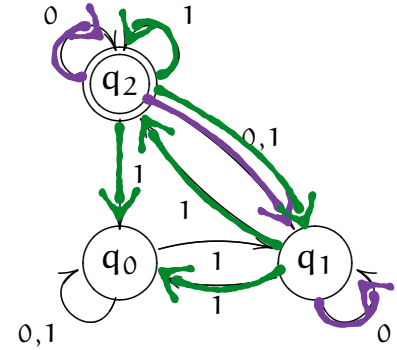
		0	1
\emptyset	\emptyset	\emptyset	\emptyset
q'_0	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
q'_1	$\{q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$
* q'_2	$\{q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q'_3	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
* q'_4	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

$\{q_0, q_1, q_2\}$

Example

NFA

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



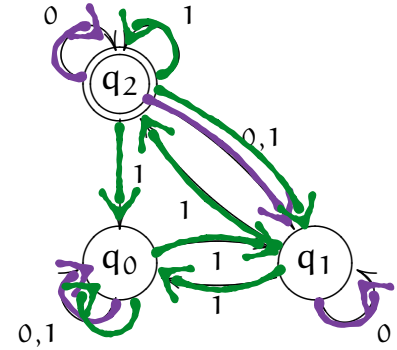
	\emptyset	0	1
\emptyset	\emptyset	\emptyset	\emptyset
$\{q_0\}$	q'_0	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	q'_1	$\{q_1\}$	$\{q_0, q_2\}$
$\{q_2\}$	* q'_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1\}$	q'_3	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	* q'_4	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$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

$\{q_0, q_1, q_2\}$

Example

NFA

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

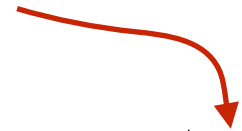
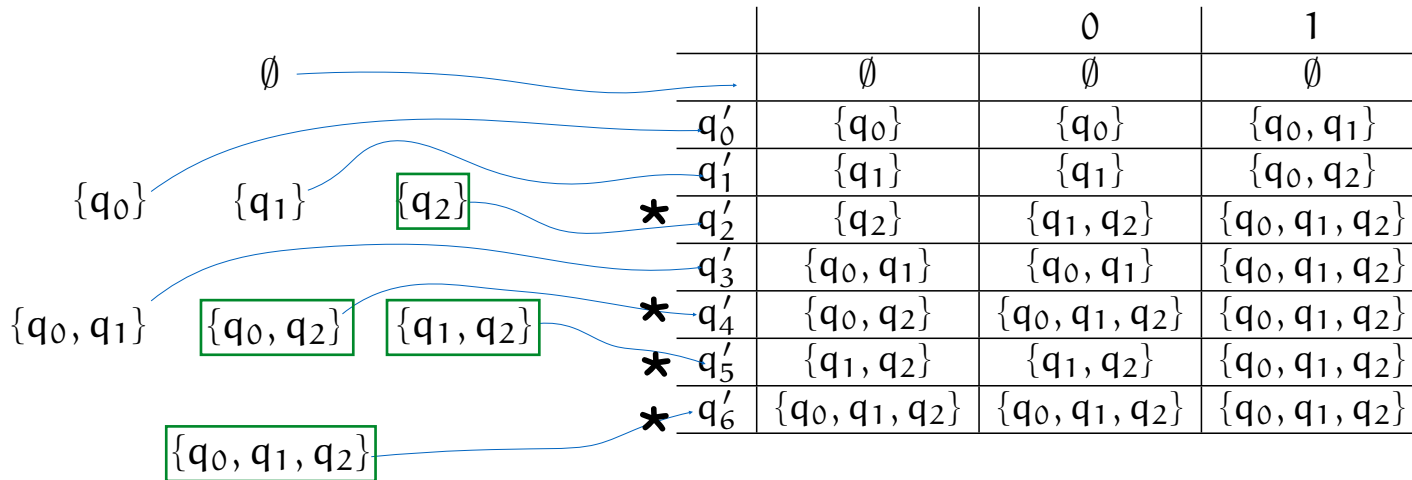
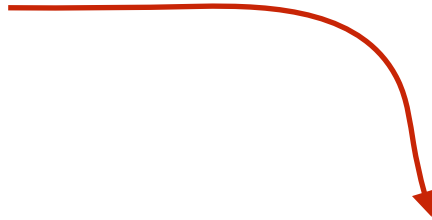


	\emptyset	0	1
\emptyset	\emptyset	\emptyset	\emptyset
$\{q_0\}$	q'_0	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	q'_1	$\{q_1\}$	$\{q_0, q_2\}$
$\{q_2\}$	* q'_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1\}$	q'_3	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	* q'_4	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$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

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 }



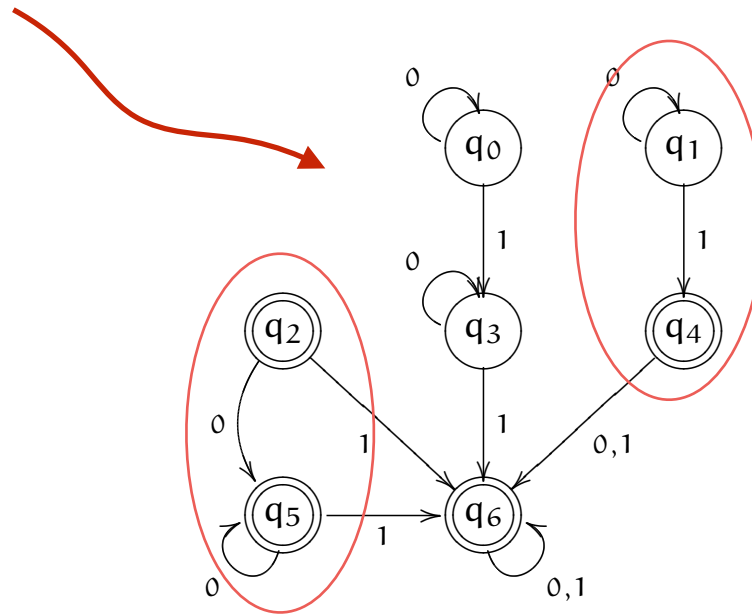
	0	1
q ₀	q ₀	q ₃
q ₁	q ₁	q ₄
* q ₂	q ₅	q ₆
q ₃	q ₃	q ₆
* q ₄	q ₆	q ₆
* q ₅	q ₅	q ₆
* q ₆	q ₆	q ₆

DFA

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



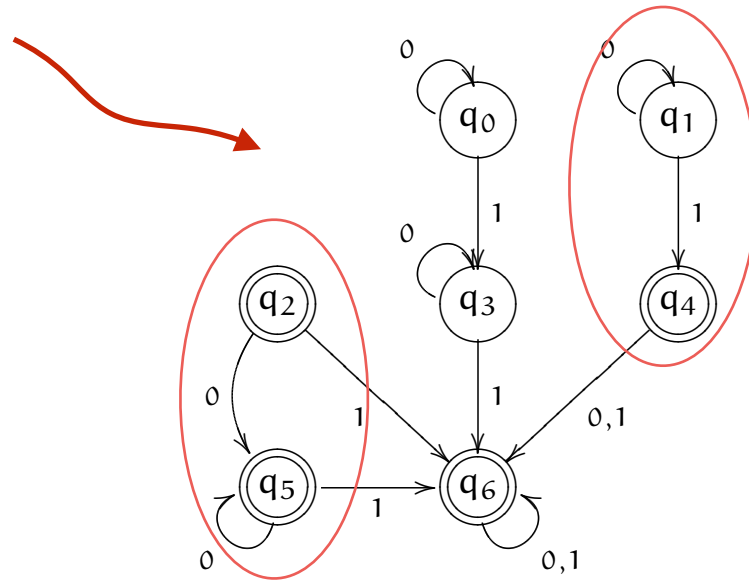
DFA with unreachable states

Example

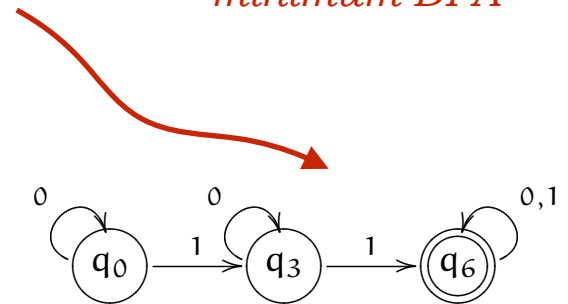
DFA

	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



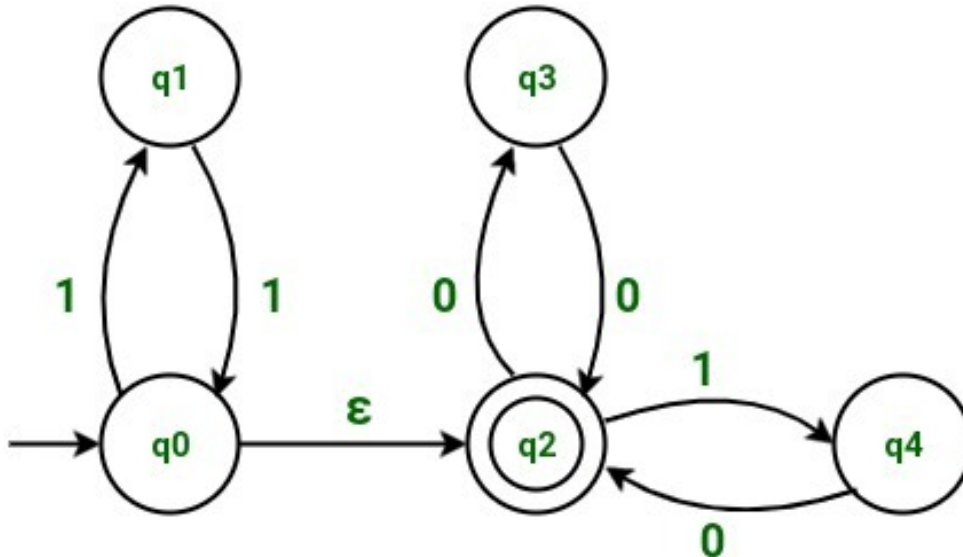
The ϵ -NFA: NFA with epsilon transitions

- Extension of finite automaton.
- The new feature: we allow transition on ϵ , the empty string.
- An NFA that is allowed to make transition spontaneously, without receiving any input symbol.
- As in the case of NFA w.r.t. DFA **this new feature does not expand the class of languages 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 -closure for extending δ to Strings

We need to define the ϵ -closure that applied to a state gives all the states reachable with ϵ -transitions



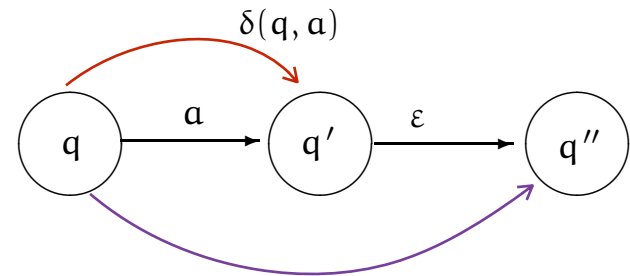
$$\epsilon\text{-closure}(q) = \{q\} \quad \epsilon\text{-closure}(q') = \{q', q''\}$$

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

The extension of δ to strings

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

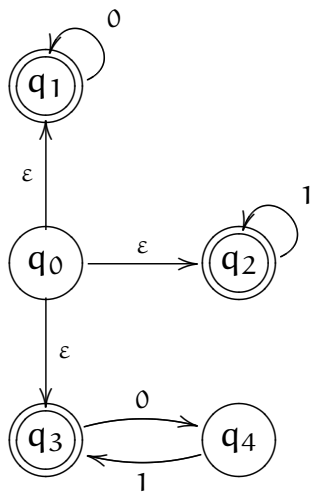
$$\left\{ \begin{array}{l} \hat{\delta}(q, \varepsilon) = \varepsilon\text{-closure}(q) \\ \hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \varepsilon\text{-closure}(\delta(p, a)) \end{array} \right.$$



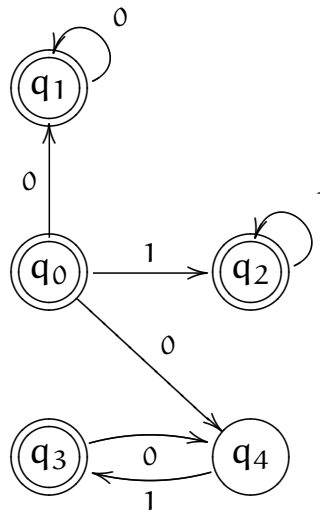
$$\hat{\delta}(q, a) = \bigcup_{p \in \hat{\delta}(q, \varepsilon)} \varepsilon\text{-closure}(\delta(p, a)) = ???$$

Example

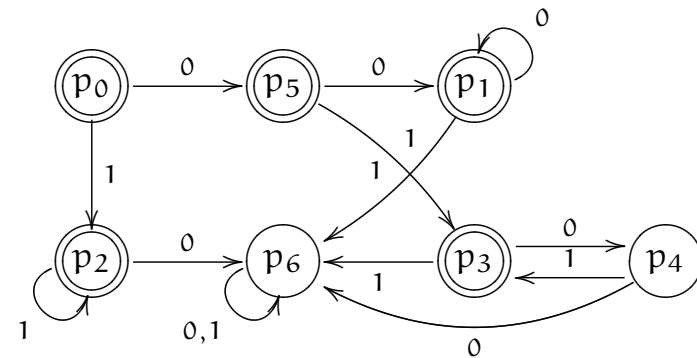
ϵ -NFA



NFA

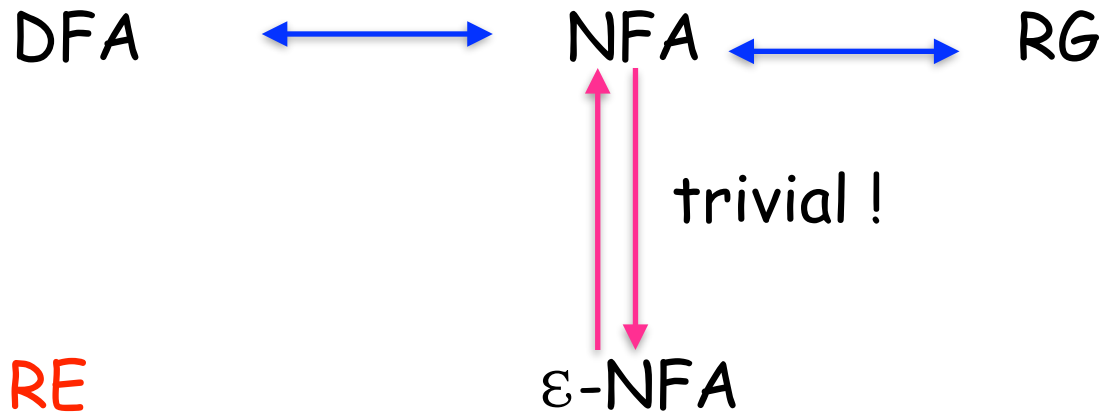


DFA



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

Roadmap: equivalence between NFA and ϵ -NFA



From ε -NFA to NFA

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

Given an

$$\varepsilon\text{-NFA } E = (Q, \Sigma, \delta_E, q_0, F_E)$$

we build a

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

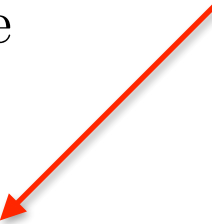
such that

$$L(E) = L(N)$$

Equivalence between ϵ -NFA and NFA

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

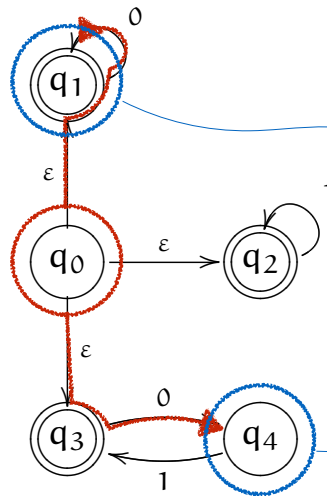
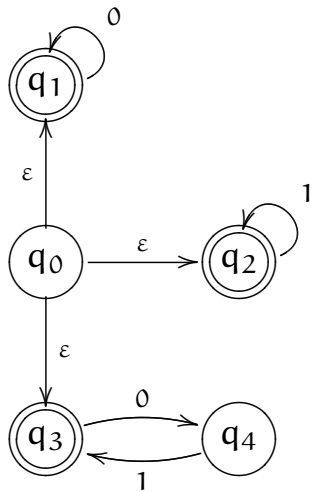
$$F_N = \begin{cases} F_E \cup \{q_0\} & \text{if } \epsilon\text{-closure}(q_0) \cap F_E \neq \emptyset \\ F_E & \text{otherwise} \end{cases}$$



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

Example

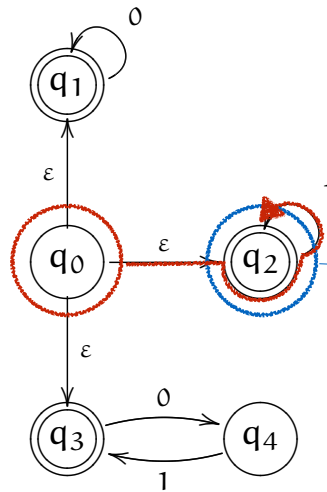
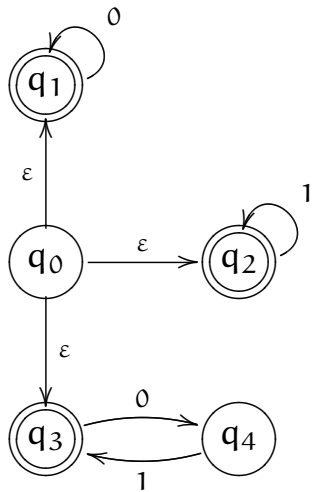
ϵ -NFA



	0	1
q0	{q1, q4}	{q2}
q1	{q1}	\emptyset
q2	\emptyset	{q2}
q3	{q4}	\emptyset
q4	\emptyset	{q3}

Example

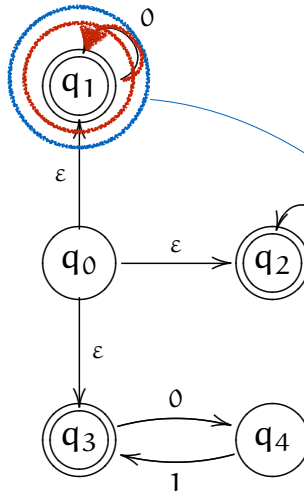
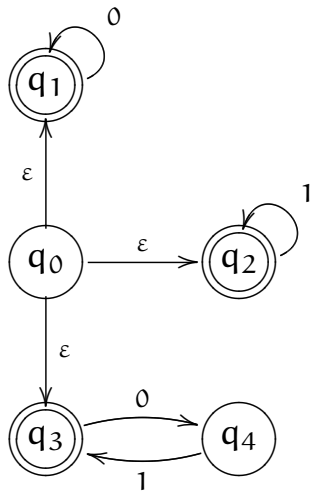
ϵ -NFA



	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

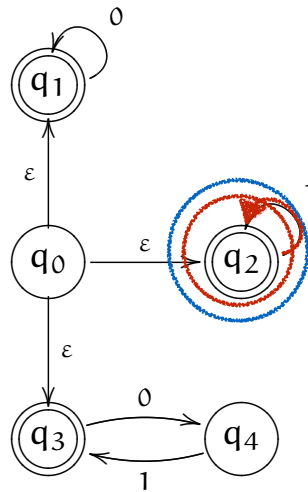
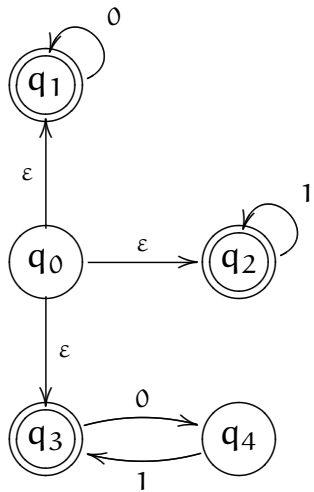
ϵ -NFA



	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

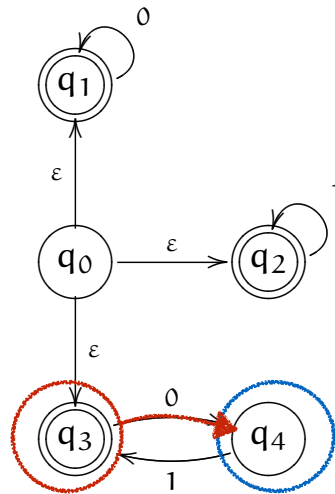
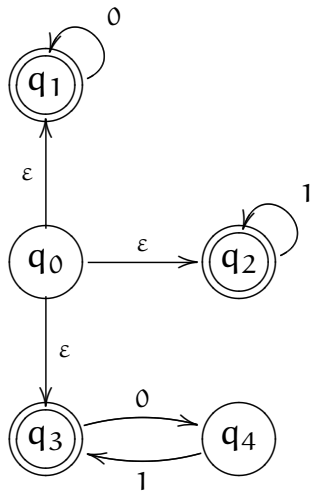
ϵ -NFA



	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

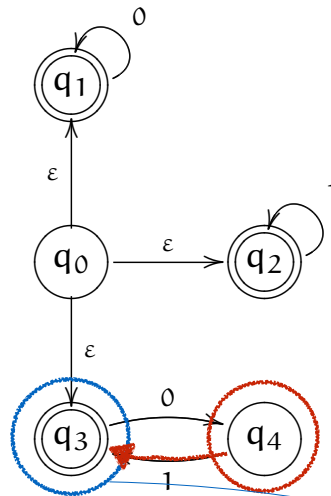
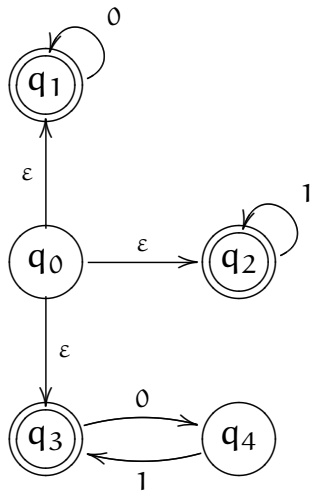
ϵ -NFA



	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	$\{q_3\}$

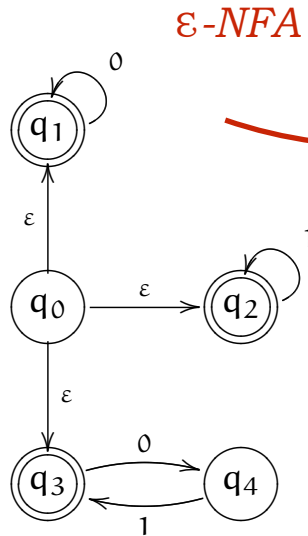
Example

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	$\{q_3\}$

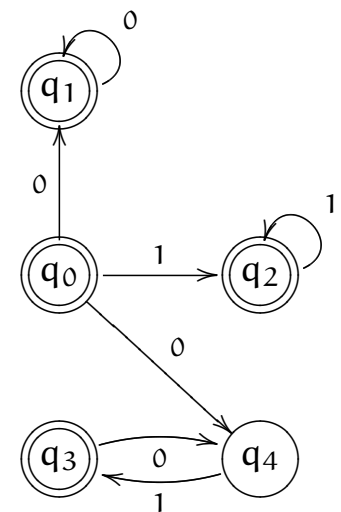
Example



NFA

	0	1
q0	{q1, q4}	{q2}
q1	{q1}	\emptyset
q2	\emptyset	{q2}
q3	{q4}	\emptyset
q4	\emptyset	{q3}

NFA



Operations on languages: recap.

Union: $A \cup B$

Intersection: $A \cap B$

Difference: $A \setminus B$

Complement: $\text{compl}(A) = \Sigma^* - A$

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

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

Regular Expressions

A **regular expression** denotes a **set** of strings (a language).

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

- \emptyset denoting the **empty set**,
- ϵ denoting the set $\{\epsilon\}$,
- a in Σ denoting the set containing only the character $\{a\}$

If r and s are regular expression (denoting the sets R and S , respectively) then $(r+s)$, (rs) and r^* denotes the set $R \cup S$, RS and R^* , respectively.

$L(r)$ indicates the language denoted by r

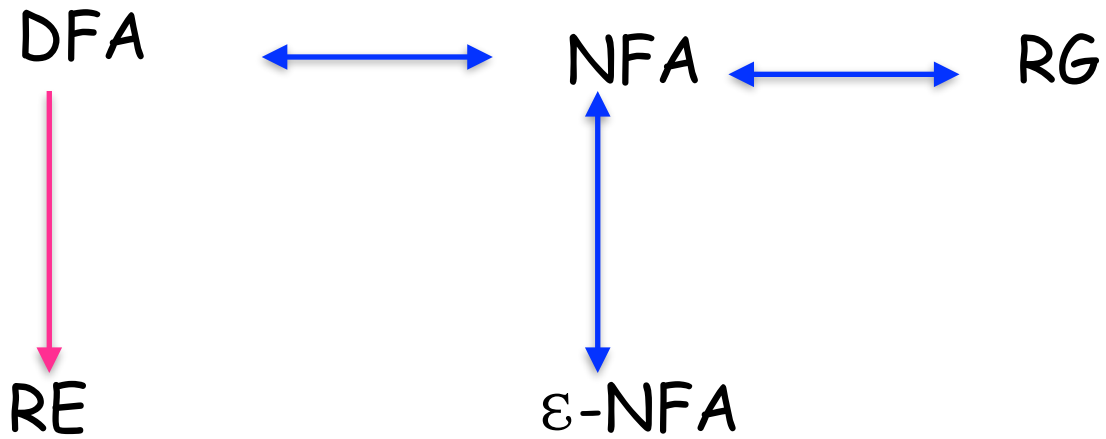
Examples

- $(0^* + 1^* + (01)^*)$ denotes the language

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

- $a|b^*$ denotes
 $\{\epsilon, "a", "b", "bb", "bbb", \dots\}$
- $(a+b)^*$ denotes
all the strings formed with "a" and "b"
- $ab^*(c+\epsilon)$ 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



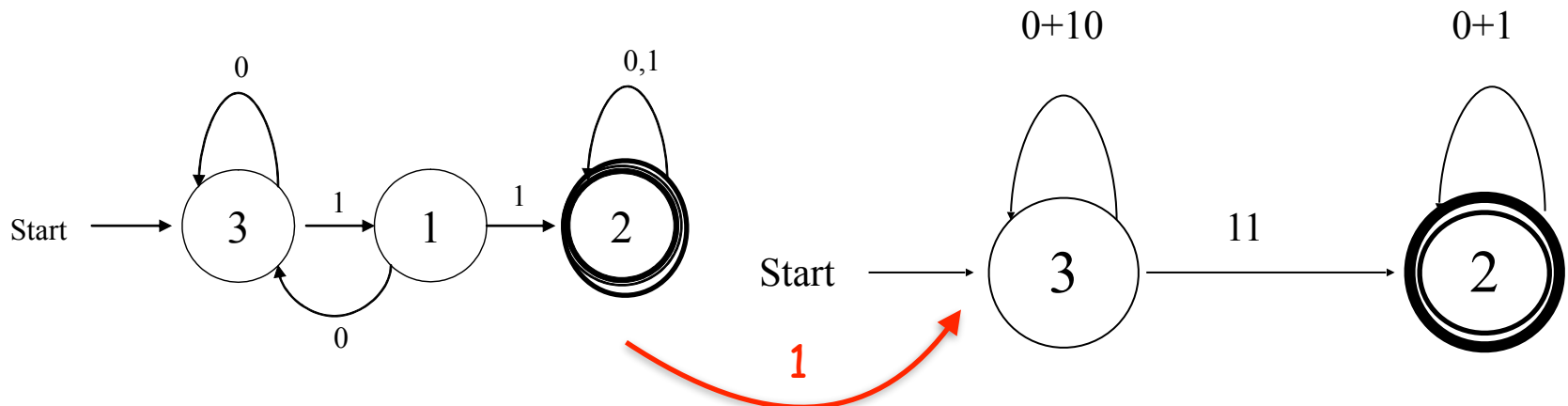
Encoding the language of a DFA into a RE

Theorem 3

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

Construction:

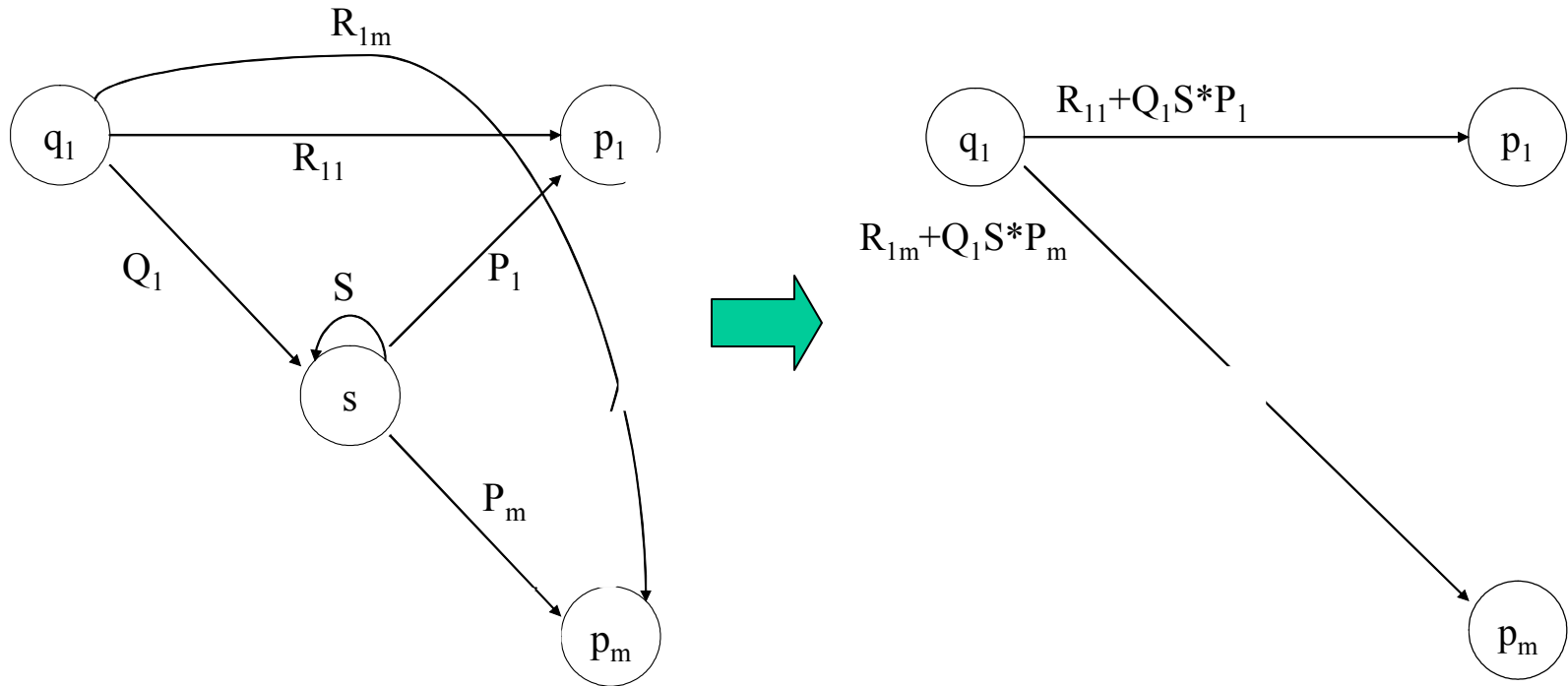
- 1) Eliminates states of the automaton replacing the edges with regular expressions that includes the behavior of the eliminated paths
- 2) When the automaton has just one starting and all final states, we synthesize the corresponding RE



State Elimination

Note: q_i and p_j may be the same state!

- Figure below shows the elimination of a state s . The labels on all edges are regular expressions.
- To remove s , we must make labels for the paths between q_1 and p_1, \dots, p_m we had in the original DFA through s .



From a DFA to RE State Elimination Point (1)

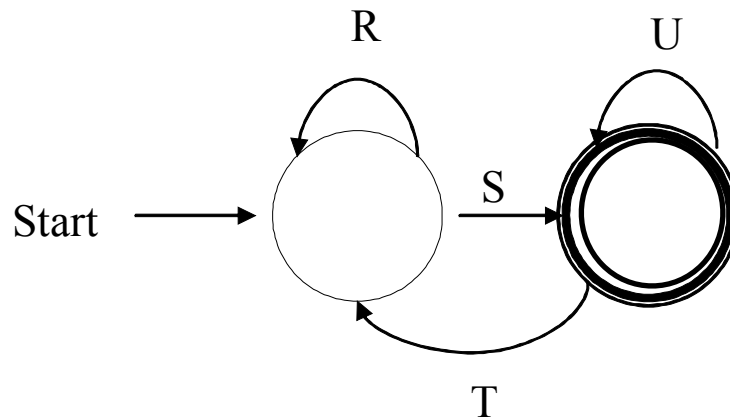
Apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges:

- Start with intermediate states and then moving to accepting states,

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

From a DFA to RE State Elimination Point (2)

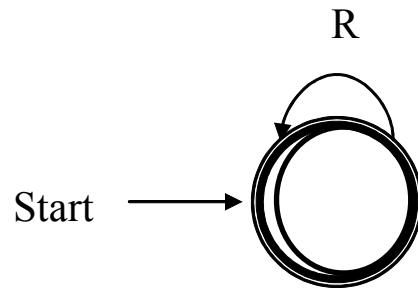
Just one final state and a different starting state



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

From a DFA to RE State Elimination Point (2)

Just one final state that coincides with the starting state



We can describe this automaton as simply R^* .

From a DFA to RE State Elimination Point (2)

Several final states s_1, s_2, \dots, s_n

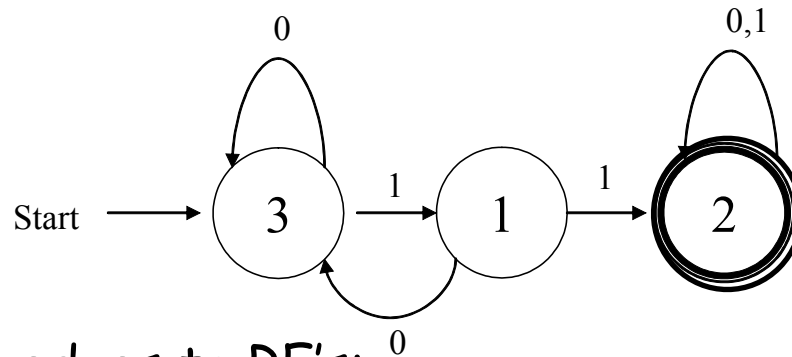
Repeat the previous steps for each s_i turning any other accepting state in non accepting.

In this way we get n different regular expressions, R_1, R_2, \dots, R_n .

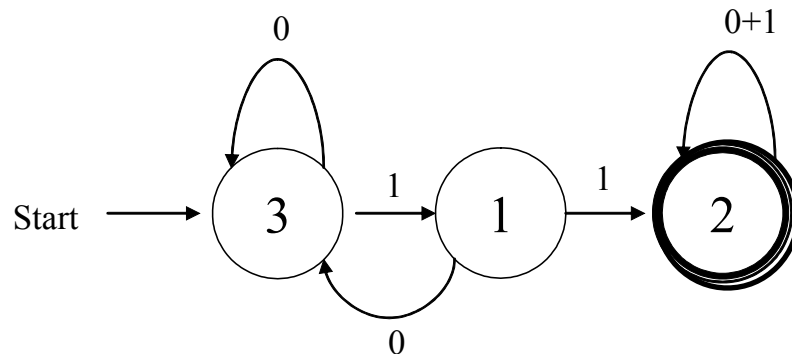
The desired regular expression for the automaton is then the union of each of the n regular expressions: $R_1 \cup R_2 \dots \cup R_N$

DFA \rightarrow RE Example

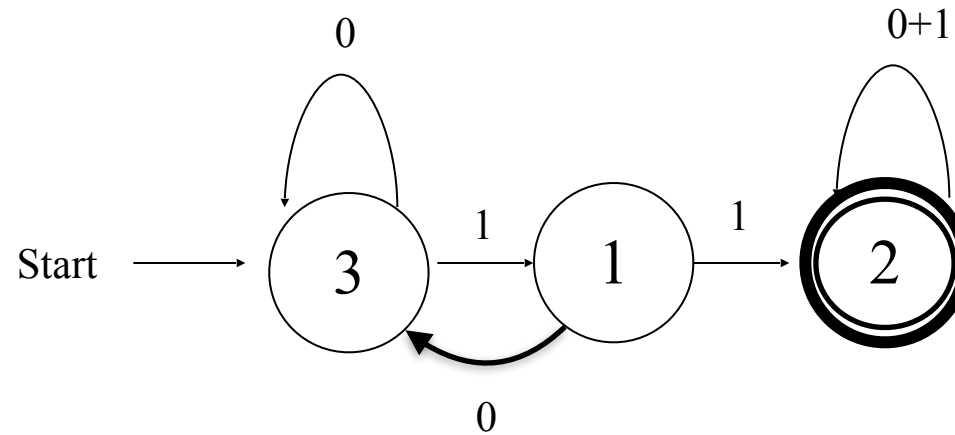
- Convert the following to a RE



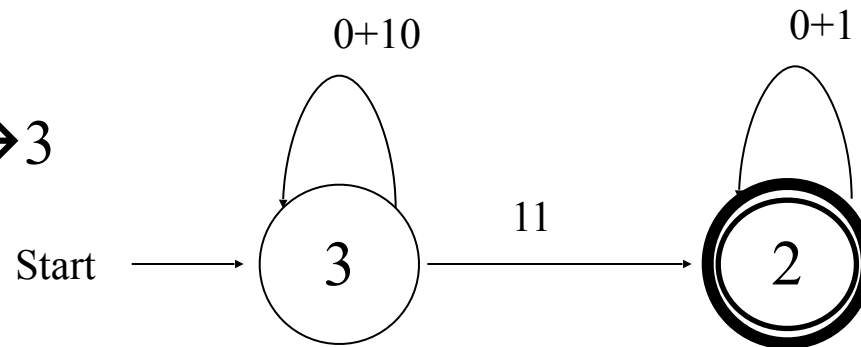
- First convert the edges to RE's:



DFA → RE Example (2)



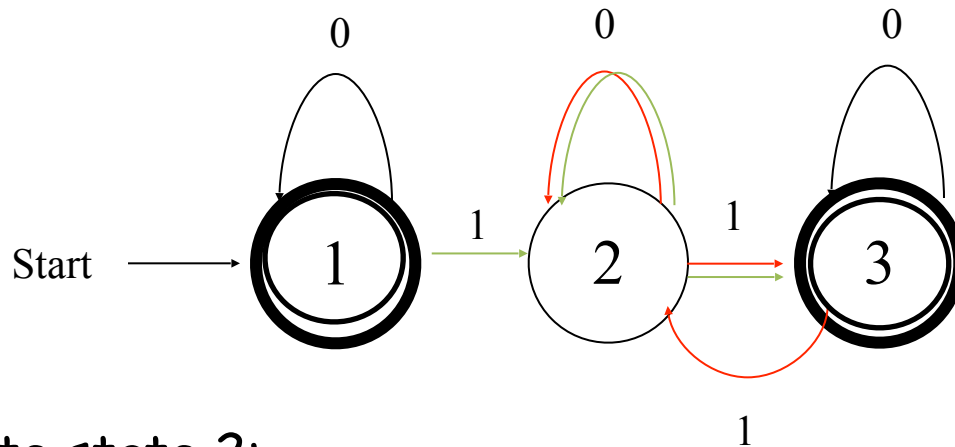
Note edge from 3 → 3



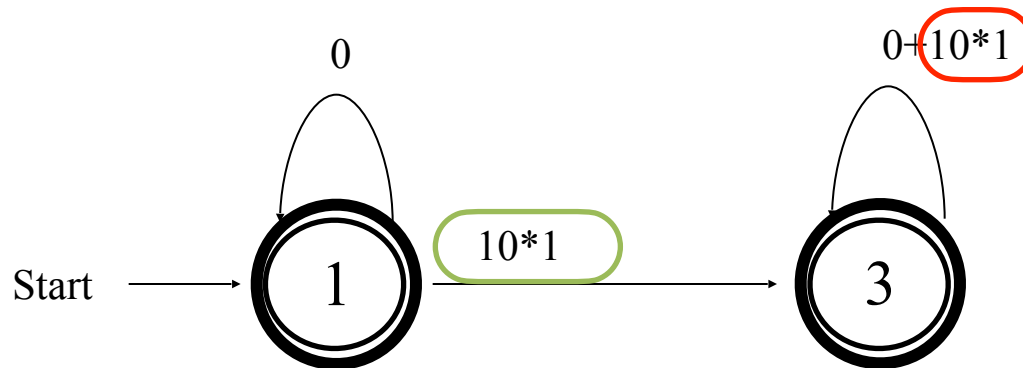
Answer: $(0+10)^*11(0+1)^*$

Third Example

- Automata that accepts even number of 1's

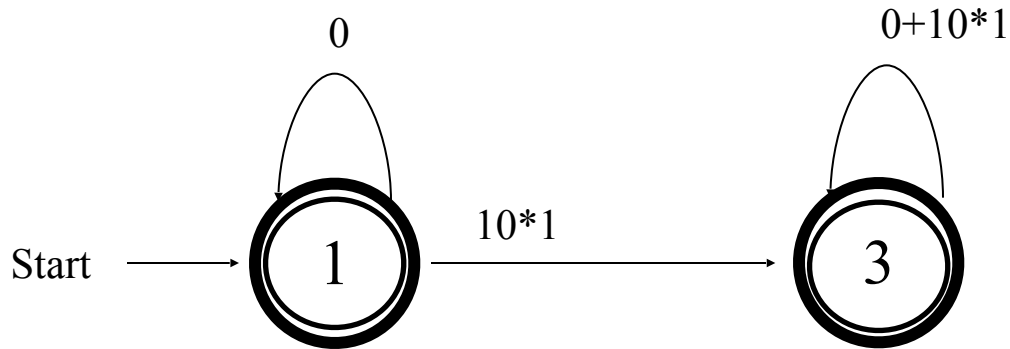


- Eliminate state 2:

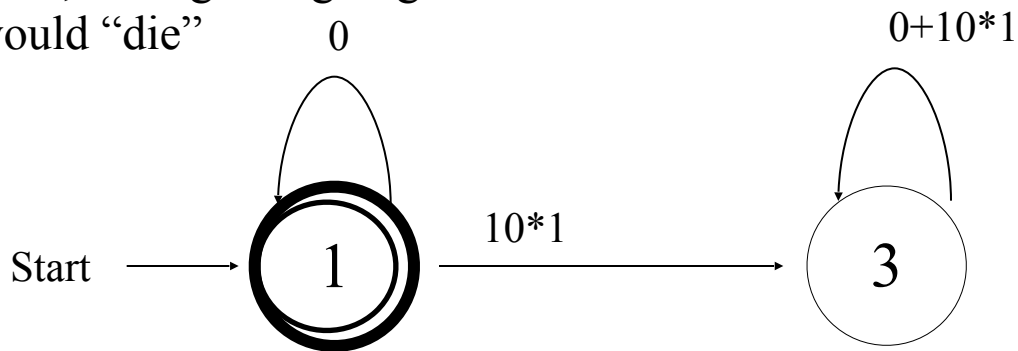


Third Example (2)

- Two accepting states, turn off state 3 first

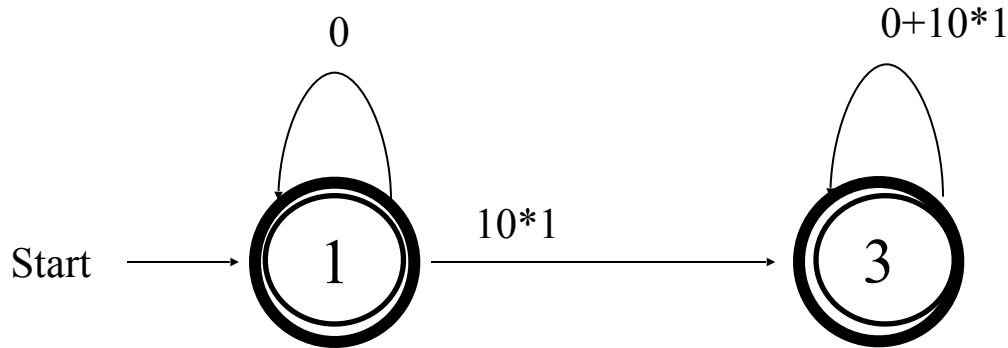


This is just 0^* ; can ignore going to state 3 since we would "die"

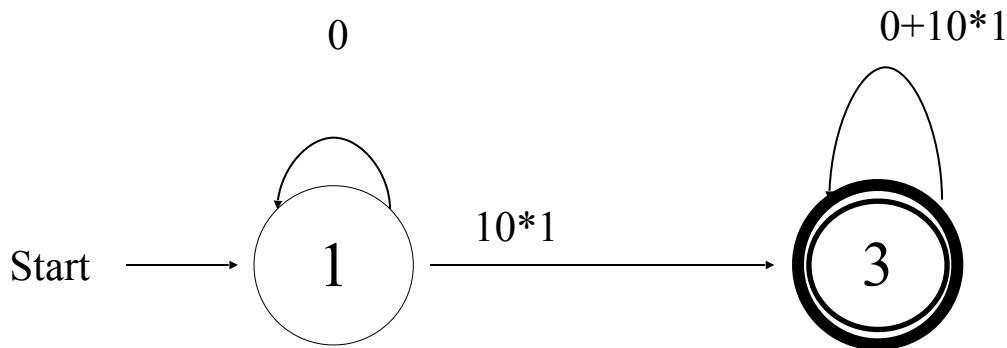


Second Example (3)

- Turn off state 1 second:



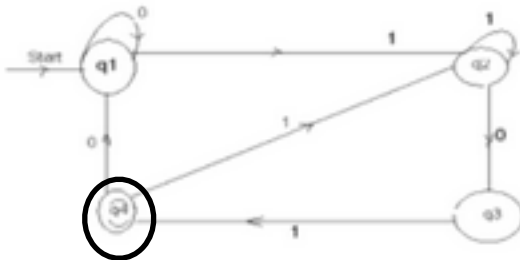
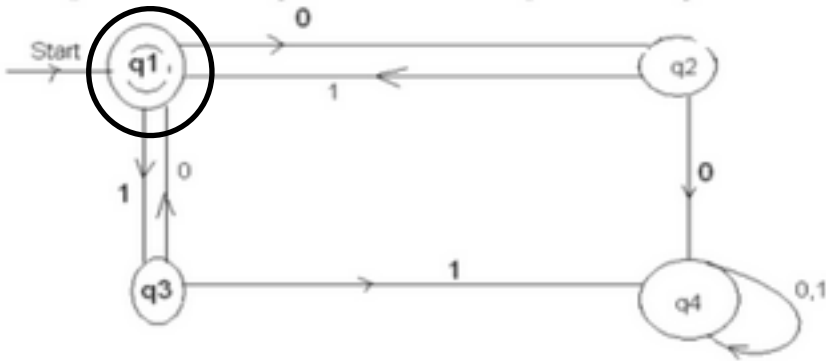
This is just $0^*10^*1(0+10^*1)^*$



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

Exercises

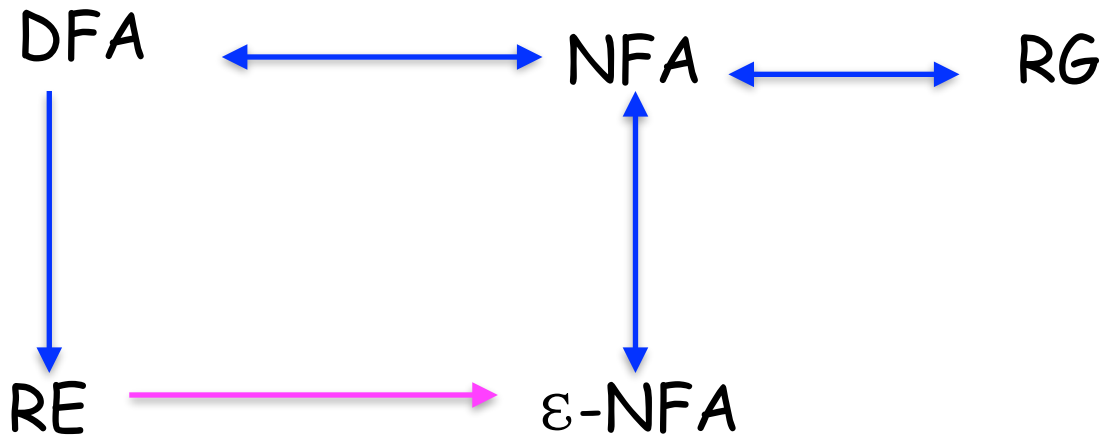
Convert the following DFA into a RE







Roadmap

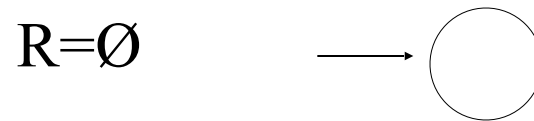
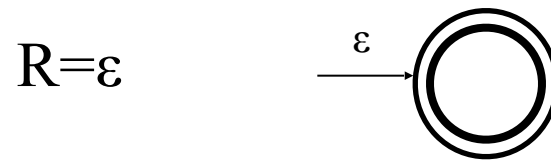
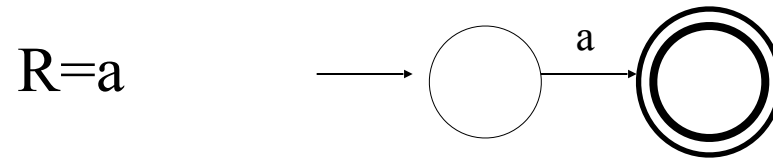


Converting a RE to an Automata

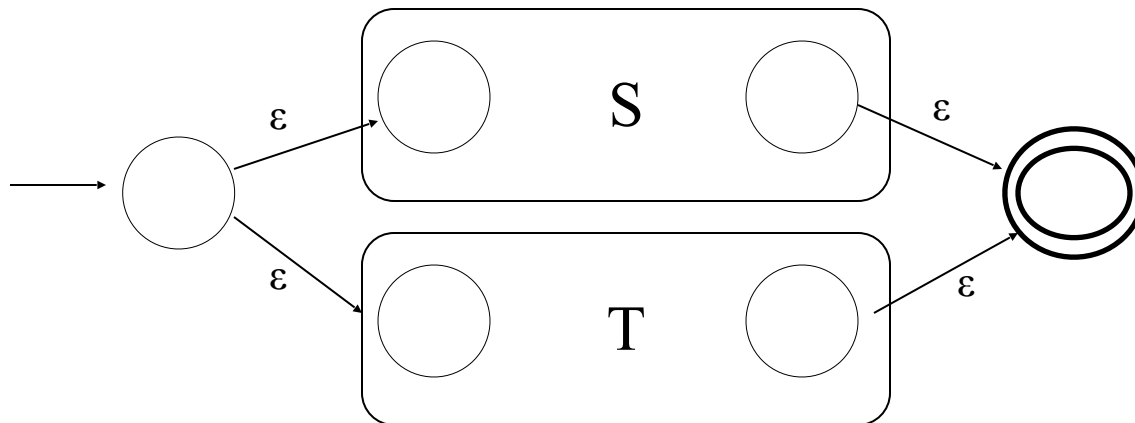
- We can convert a RE to an ϵ -NFA
 - Inductive construction
 - Start with a simple basis, use that to build more complex parts of the NFA

RE to ϵ -NFA

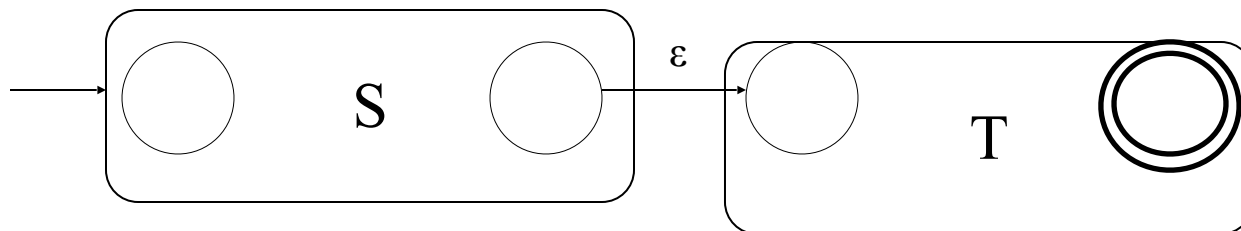
- Basis:



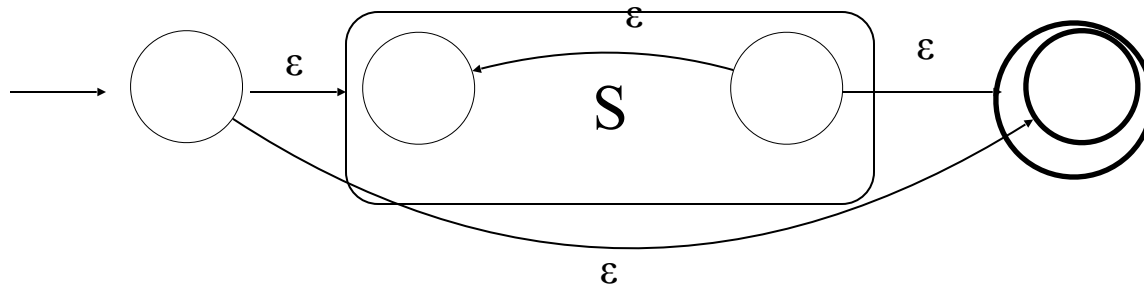
$R=S+T$



$R=ST$

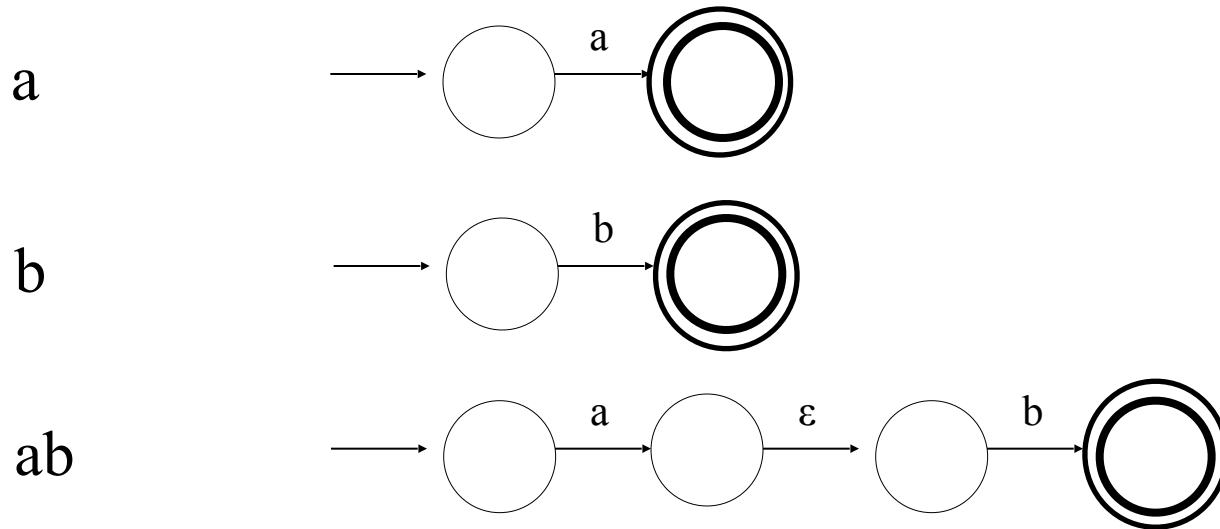


$R=S^*$



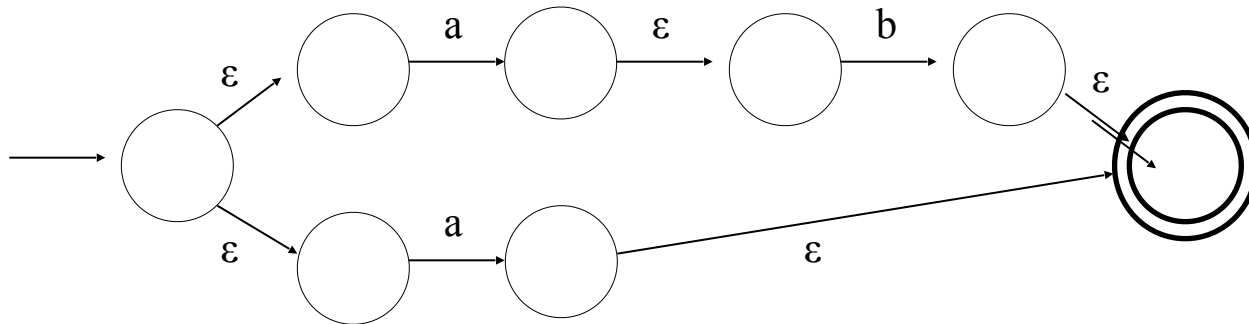
RE to ϵ -NFA Example

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

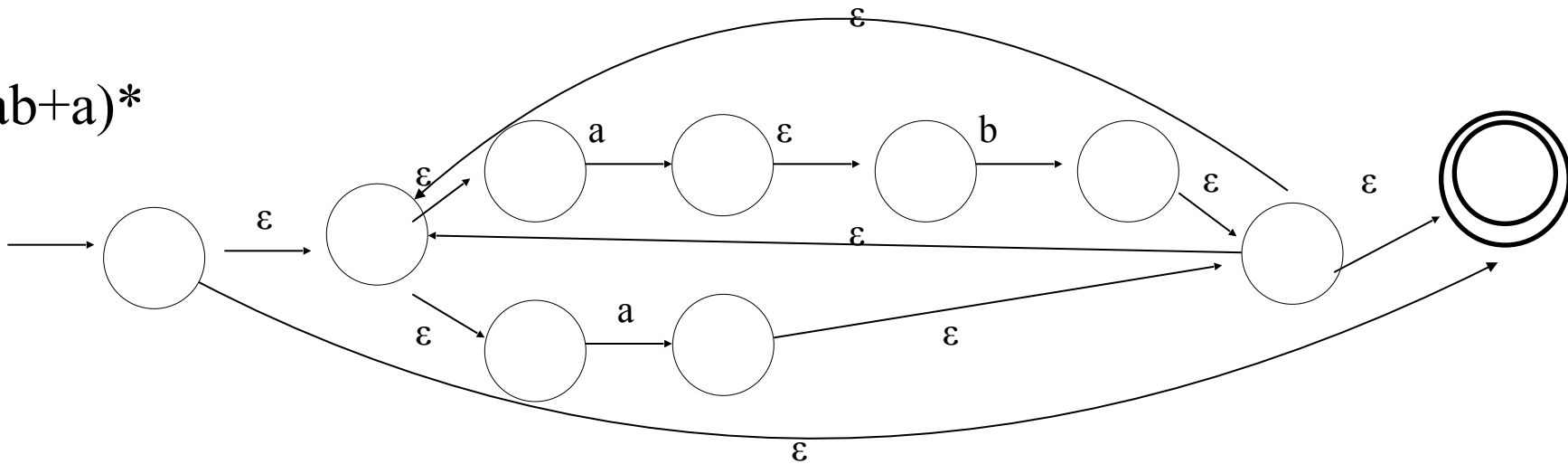


RE to ϵ -NFA Example (2)

$ab+a$



$(ab+a)^*$

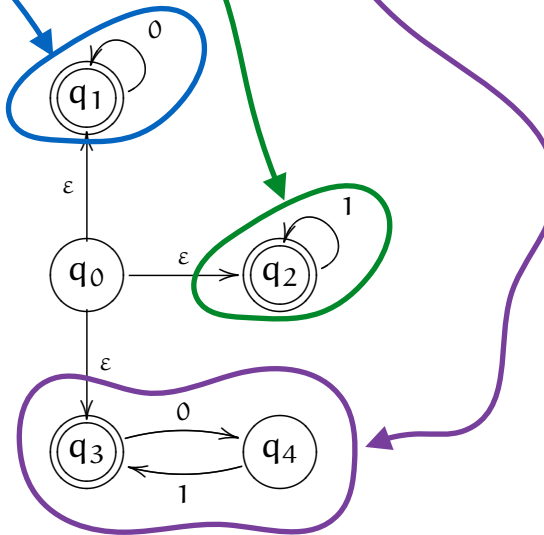


Esempio: from RE to ϵ -NFA

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

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

ϵ -
NFA



What have we shown?

- Regular expressions, finite state automata and regular grammars are different ways of expressing the same languages
- 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 1's is typically easier to design as a NFA or DFA and then convert it to a RE

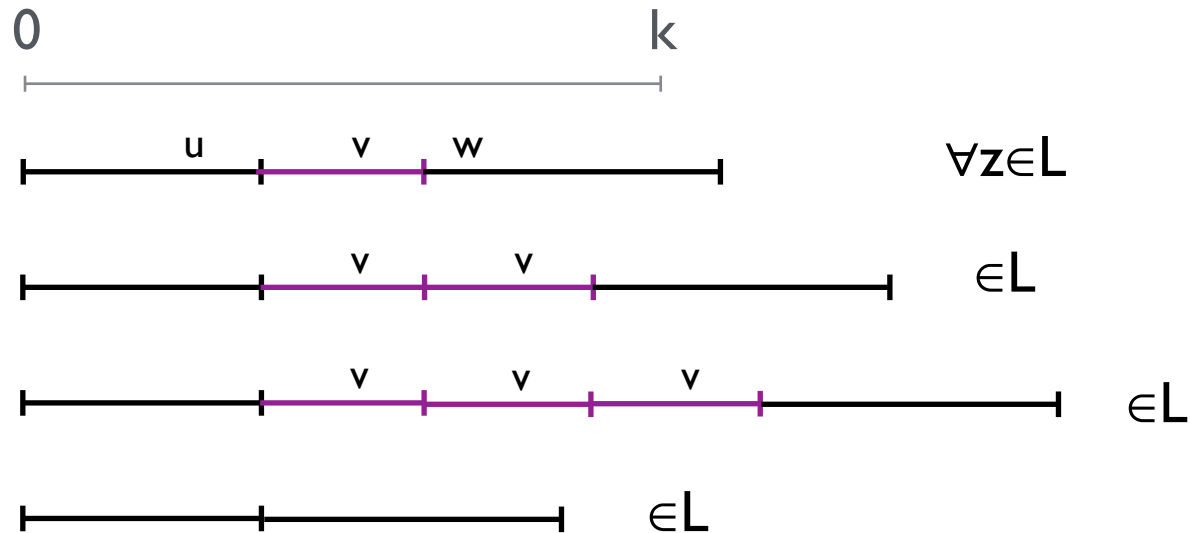
Not all languages are regular!

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

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

$z = uvw$ with $|uv| \leq k, |v| > 0$ 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 a regular language!**

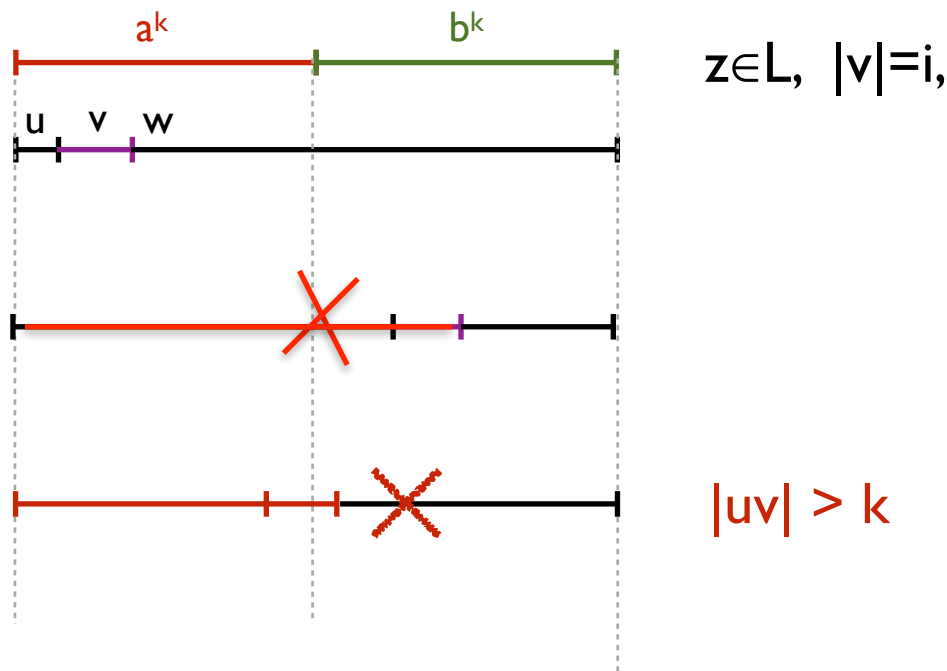
If $\forall k \in \mathbf{N} \exists z \in L. |z| \geq k$ for all possible splitting

$z = uvw$ 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!**

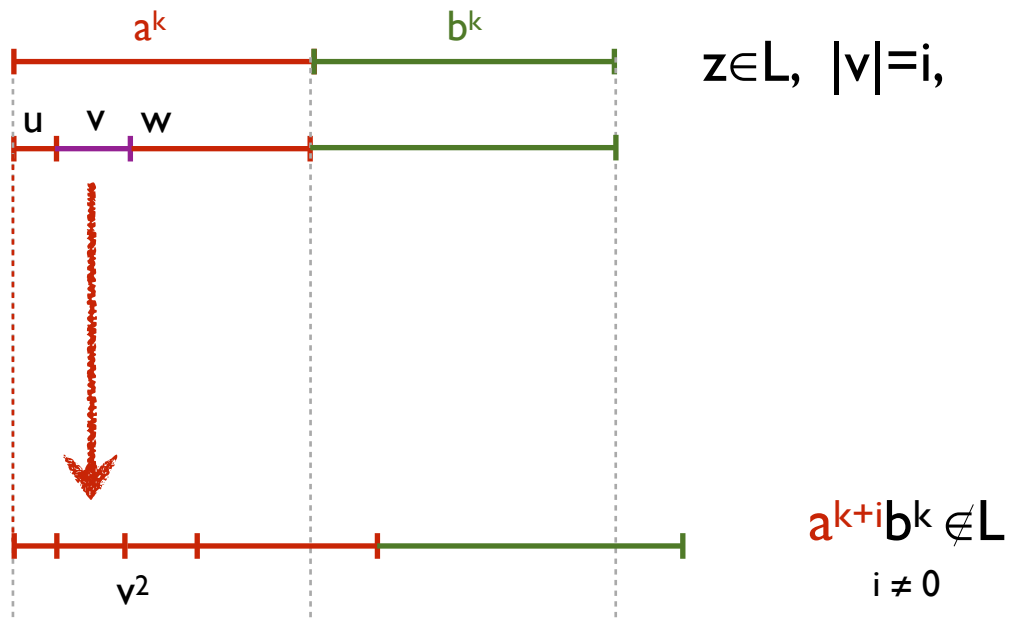
Exemple

- $L = \{ a^n b^n \mid n \in \mathbb{N} \}$, take any $k \in \mathbb{N}$
- Consider the string $z = a^k b^k$



Esempio

- $L = \{ a^n b^n \mid n \in \mathbb{N} \}$, take any $k \in \mathbb{N}$
- Consider the string $z = a^k b^k$



Exercises

Prove that the following are not regular languages

$$L_{pal} = \{w \in \{0,1\}^* \mid w = w^R\} = \{\epsilon, 0, 1, 00, 11, 00100, 01110, \dots\}$$

$$L_1 = \{0^n 1^m \mid n \leq m\}$$

$$L_2 = \{0^n \mid n \text{ e' una potenza di } 2\}$$

$$L_3 = \{w2^n \mid w \in \{0,1\}^*, n = |w|\}$$







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

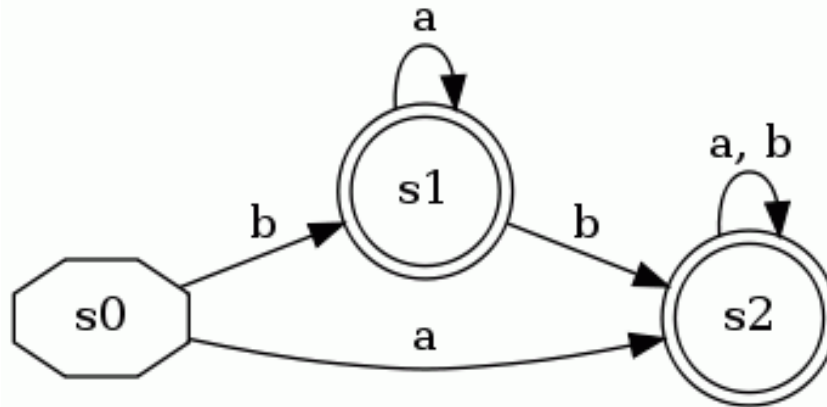
Decision Properties:

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 the DFA minimization is to automatically transform a given DFA into a state-minimized DFA
 - Several algorithms and variants are known

A DFA Minimization Algorithm

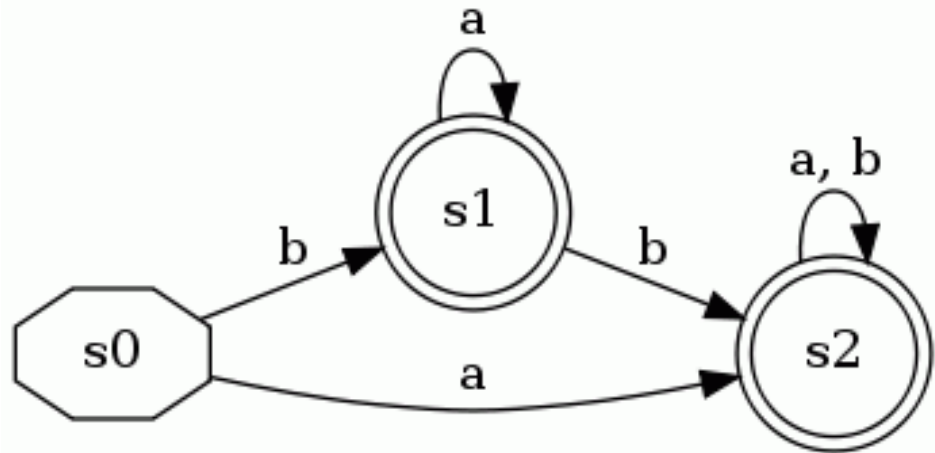
- Recall that a DFA $M=(Q, \Sigma, \delta, q_0, F)$
- Two states p and q are distinct if
 - $p \in F$ and $q \notin F$ or vice versa, or
 - $\delta(p, a)$ and $\delta(q, a)$, for some a in Σ , are distinct
- Using this inductive definition, we can calculate which states are distinct

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
 - $\text{DISTINCT}(p,q) = \varepsilon$
- Loop until no change for an iteration:
 - For every pair of states (p,q) and each symbol a
 - If $\text{DISTINCT}(p,q)$ is blank and $\text{DISTINCT}(\delta(p,a), \delta(q,a))$ is not blank
 - $\text{DISTINCT}(p,q) = a$
- Combine all states that are not distinct

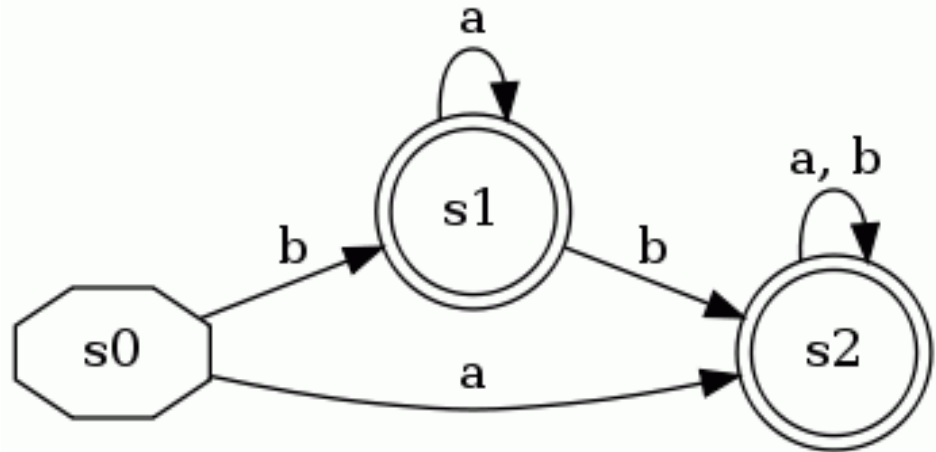
Very Simple Example

s0			
s1			
s2			
	s0	s1	s2



Very Simple Example

s0			
s1	ϵ		
s2	ϵ		
	s0	s1	s2

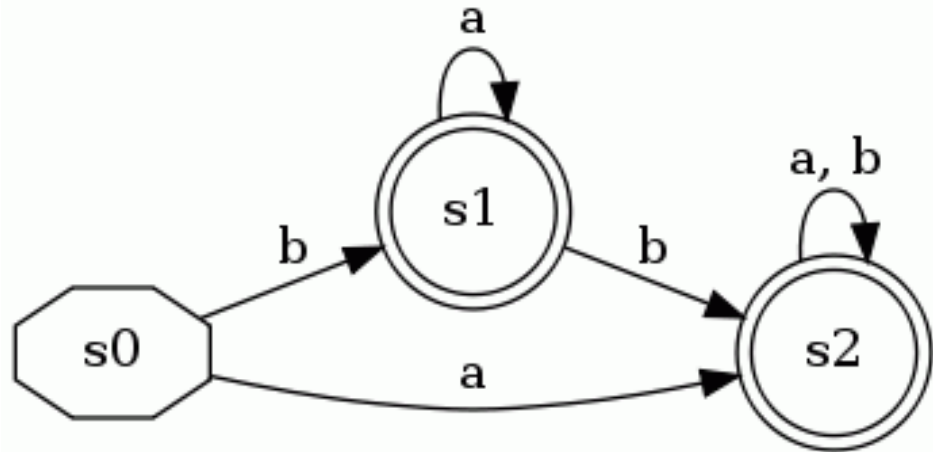


Label pairs with ϵ where one is a final state and the other is not

Very Simple Example

- $\text{DISTINCT}(p, q)$ is blank and $\text{DISTINCT}(\delta(p, a), \delta(q, a))$ is not blank
- $\text{DISTINCT}(p, q) = a$

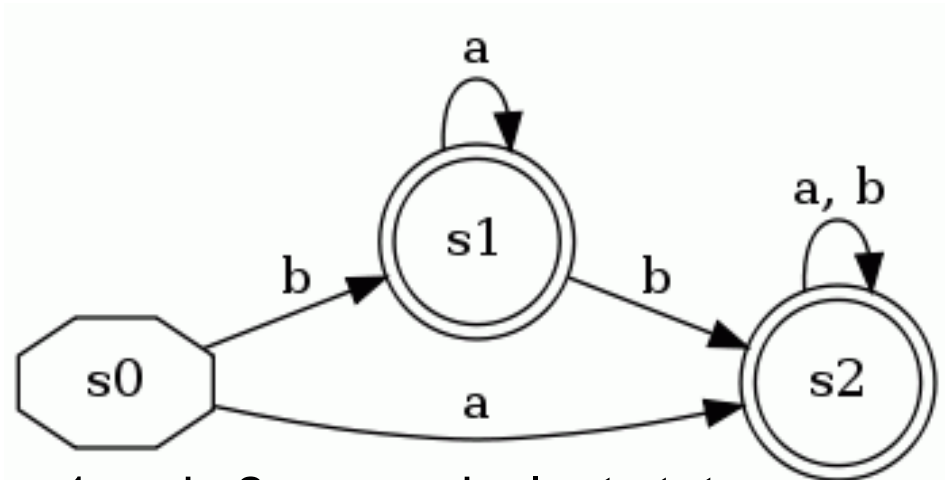
s0			
s1	ϵ		
s2	ϵ		
	s0	s1	s2



Main loop (no changes occur)

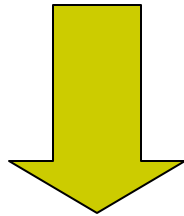
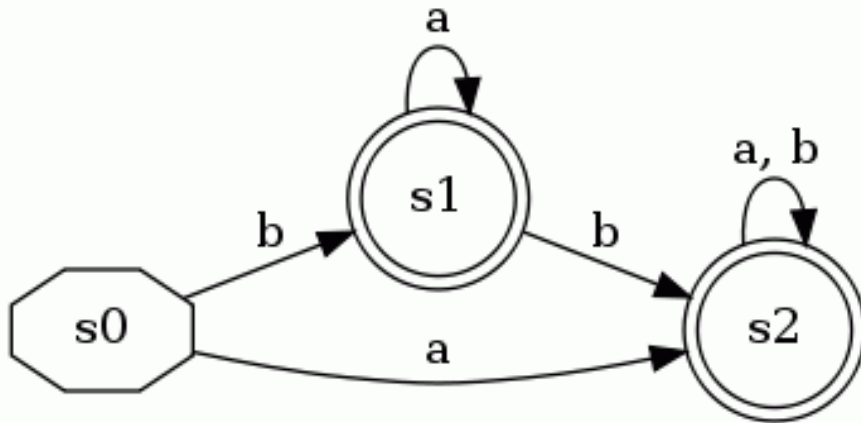
Very Simple Example

s0			
s1	ϵ		
s2	ϵ		
	s0	s1	s2

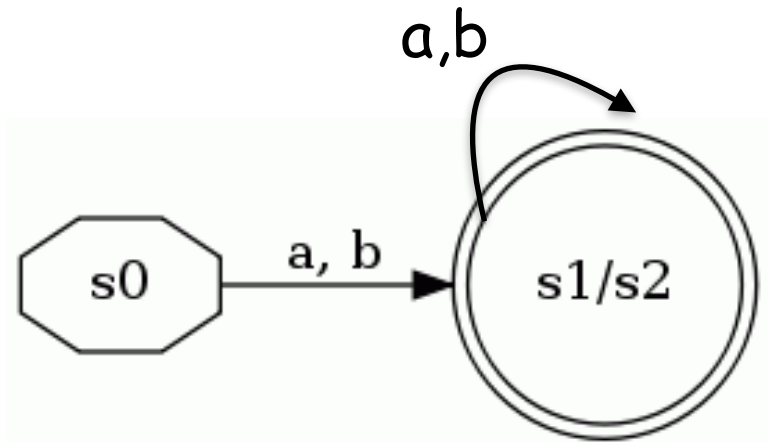
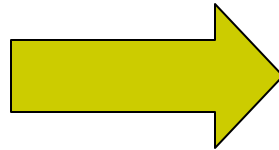


$\text{DISTINCT}(s1, s2)$ is empty, so s1 and s2 are equivalent states

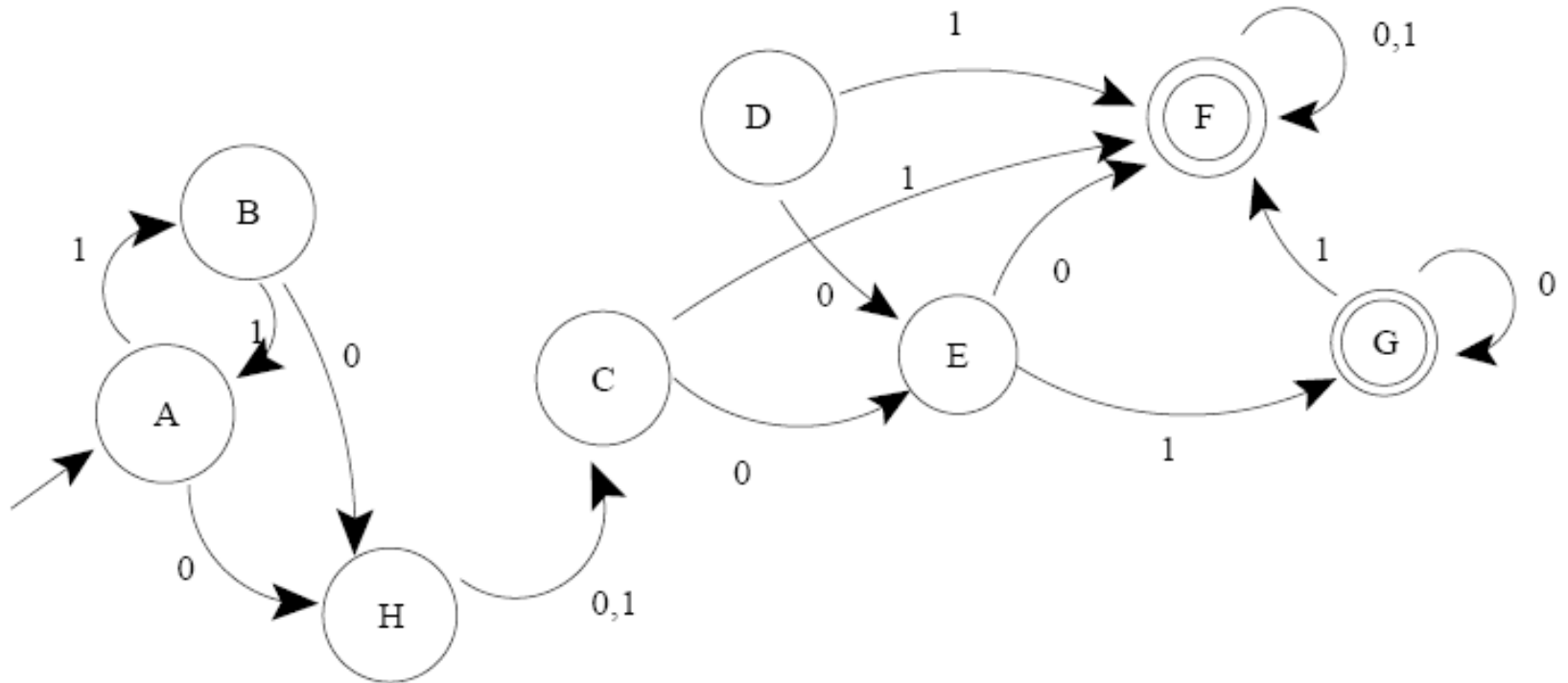
Very Simple Example



Merge s_1 and s_2



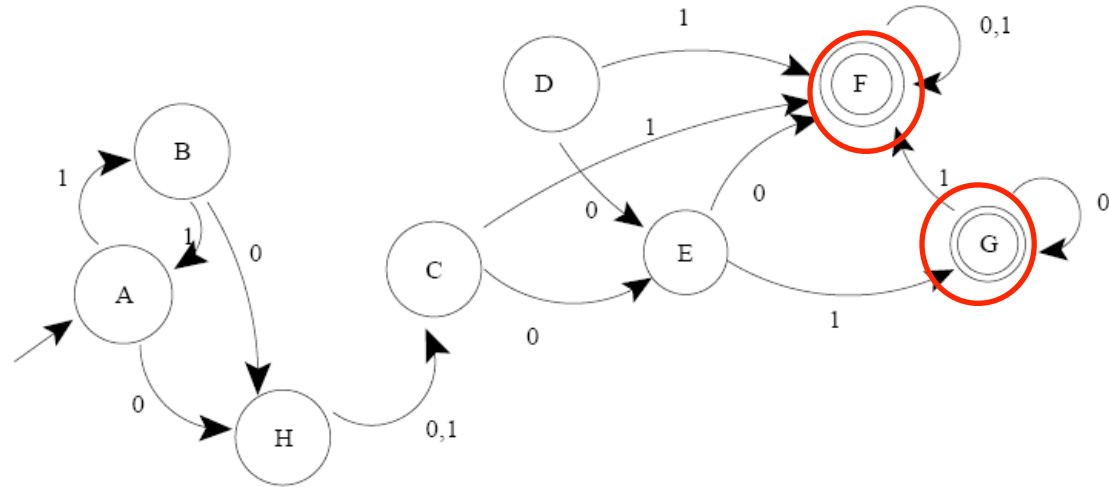
More Complex Example



More Complex Example

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

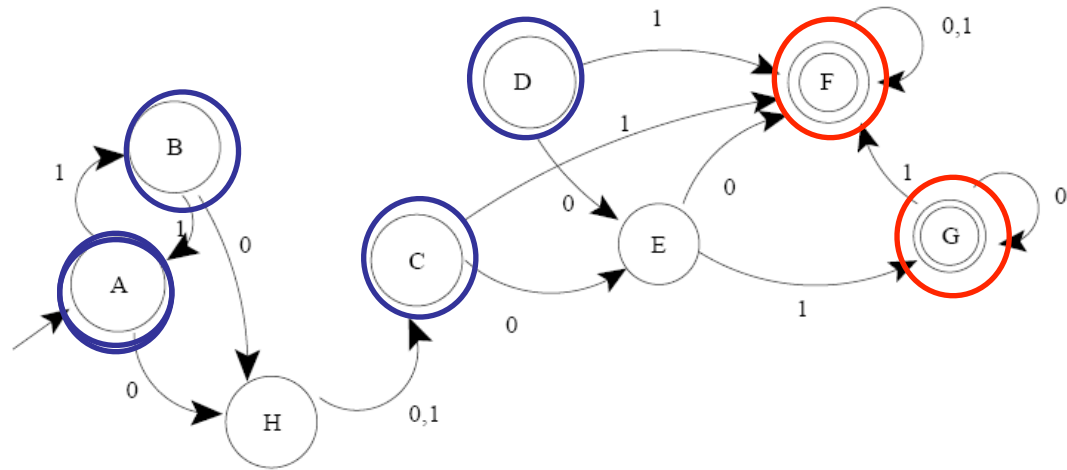
b								
c								
d								
e								
f	ε	ε	ε	ε	ε			
g	ε	ε	ε	ε	ε			
h						ε	ε	
	a	b	c	d	e	f	g	



More Complex Example

- First iteration of main ' .

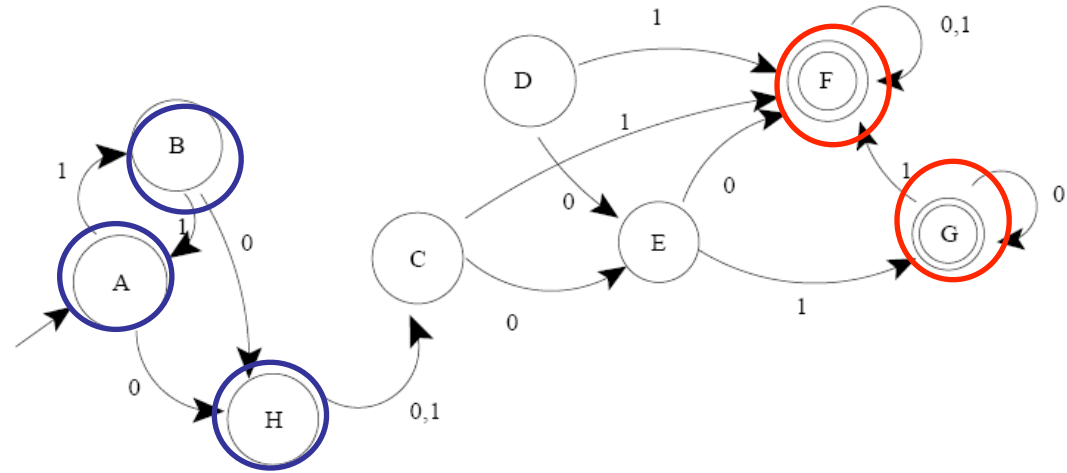
b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h			1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g



More Complex Example

- 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



More Complex Example

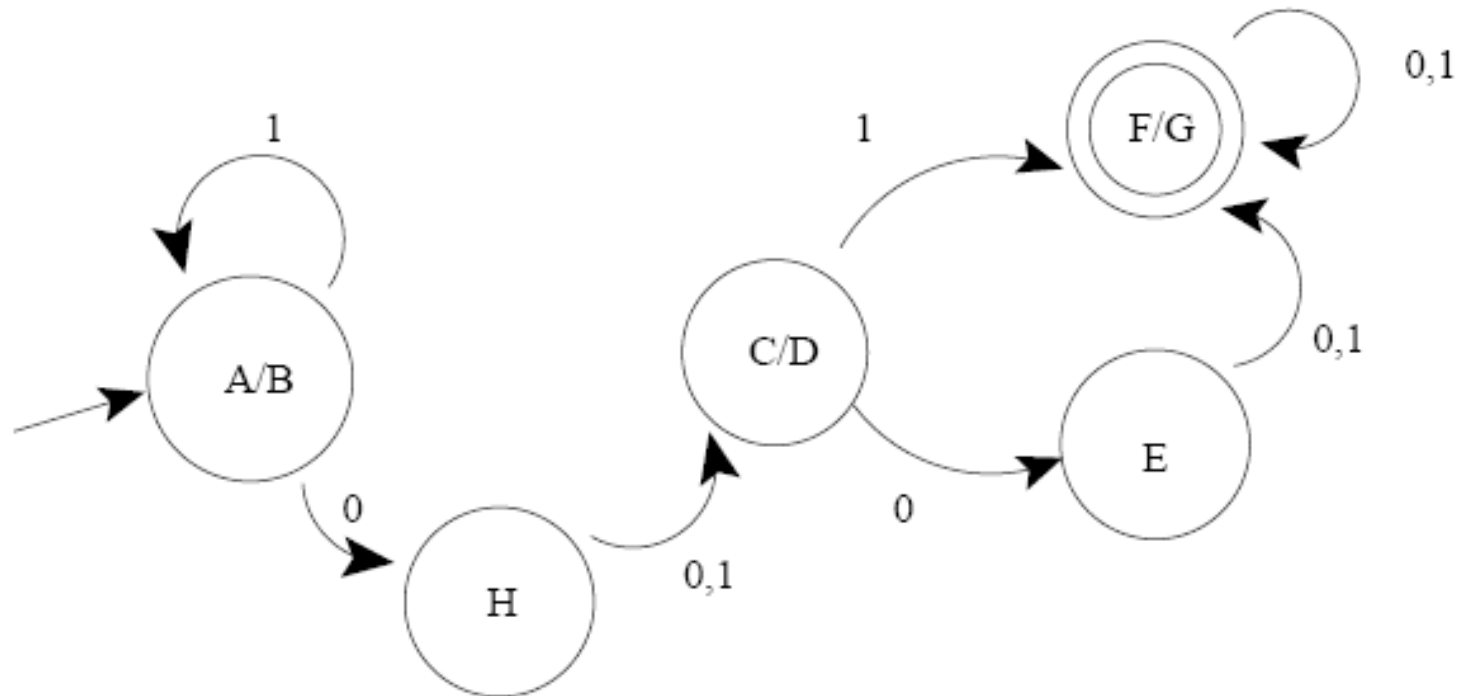
- Third iteration makes no changes
 - Blank cells are equivalent pairs of states

The table below shows a state transition matrix. The rows are labeled b, c, d, e, f, g, h and the columns are labeled a, b, c, d, e, f, g. The cells contain values 0, 1, or ϵ . Three red arrows originate from a common point above the right side of the table and point to blank cells at (b, a), (d, d), and (g, 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	1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

More Complex Example

- Combine equivalent states for minimized DFA:



Conclusion

- The algorithm described is $O(kn^2)$
 - John Hopcraft describes another more complex algorithm that is $O(k (n \log n))$

Exercises

Minimize the following automata

	0	1
→ A	B	A
B	A	C
C	D	B
* D	D	A
E	D	F
F	G	E
G	F	G
H	G	H

	0	1
→ A	B	E
B	C	F
* C	D	H
D	E	H
E	F	I
* F	G	B
G	H	B
H	I	C
* I	A	E





Linguaggi Context Free

Context free Grammars

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

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

$$U \in N \text{ and } V \in (\Sigma \cup N)^+$$

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

Example

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

$E \mapsto 0$

$E \mapsto 1$

$E \mapsto (E \text{ or } E)$

$E \mapsto (E \text{ and } E)$

$E \mapsto (\text{not } E)$

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

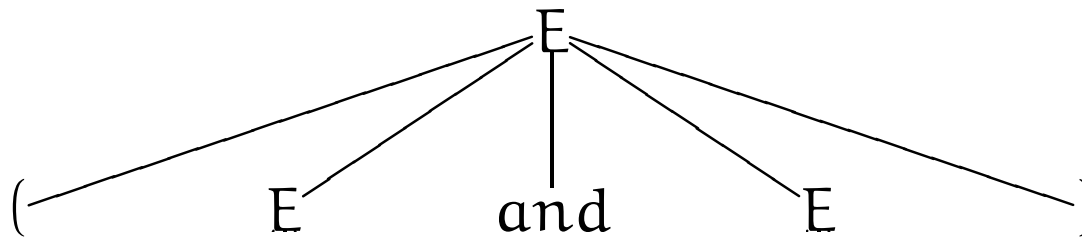
Given a grammar (Σ, N, S, P) .

The parse tree is the graph representation of a derivation, which can be defined in the following way:

- every vertex has a label in $\Sigma \cup N \cup \{\varepsilon\}$,
- the label of the root and of every internal vertex belongs to N ,
- if a vertex is labeled with A and has m children labeled with X_1, \dots, X_k
- then the production $A \rightarrow X_1 \dots X_k$ belongs to P ,
- if a vertex is labeled with ε then it is a leaf and is an only child.

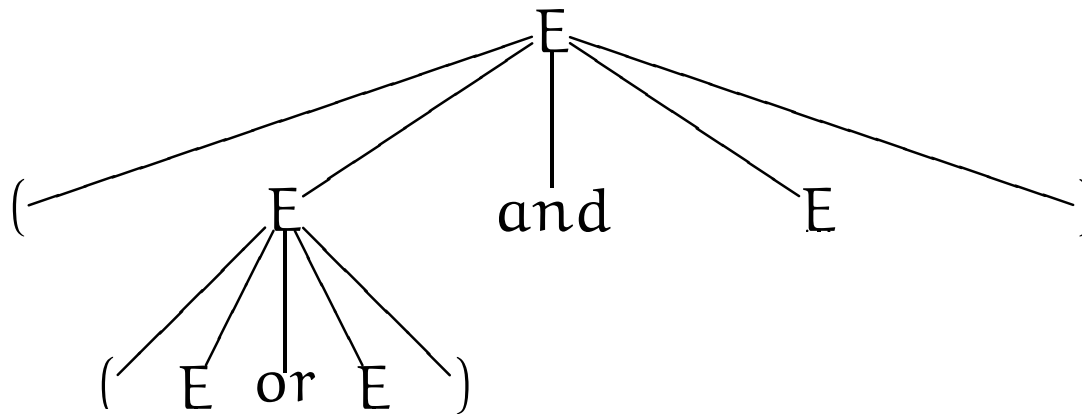
Example

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



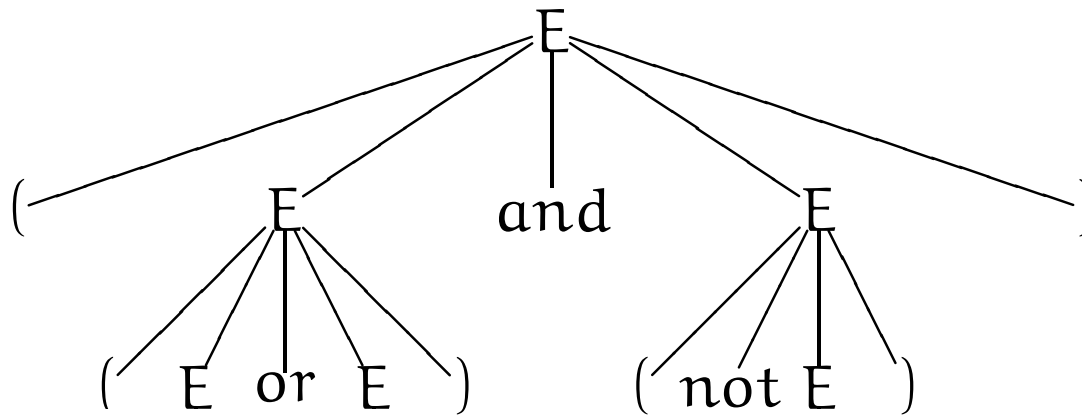
Example

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



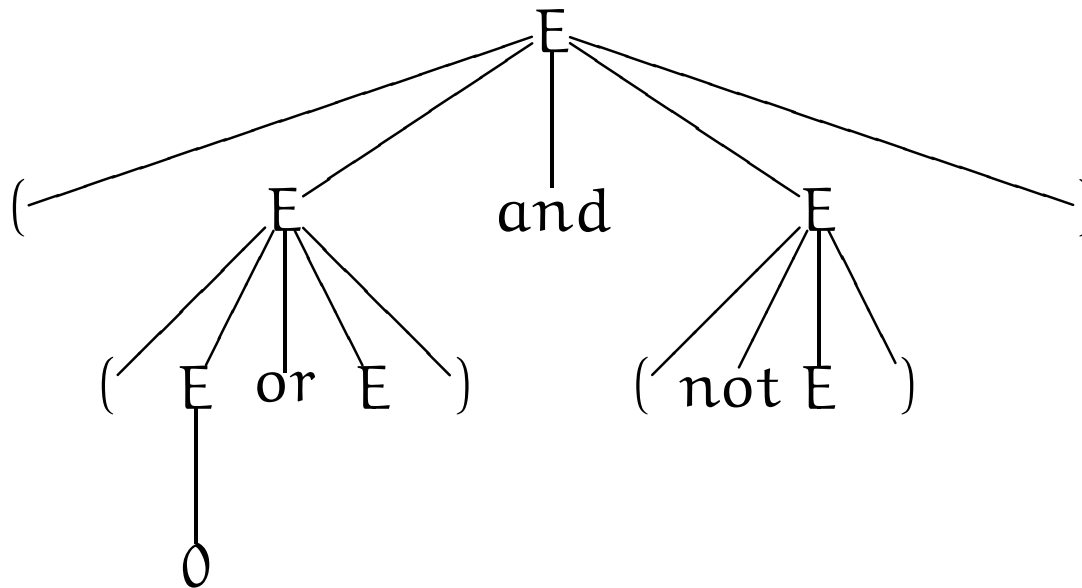
Example

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



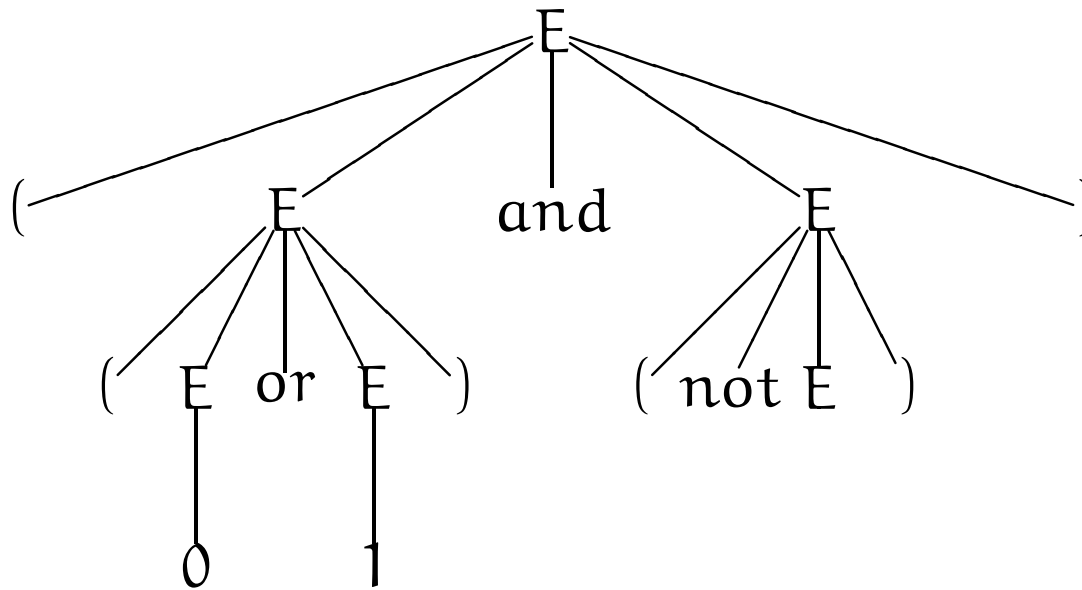
Example

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



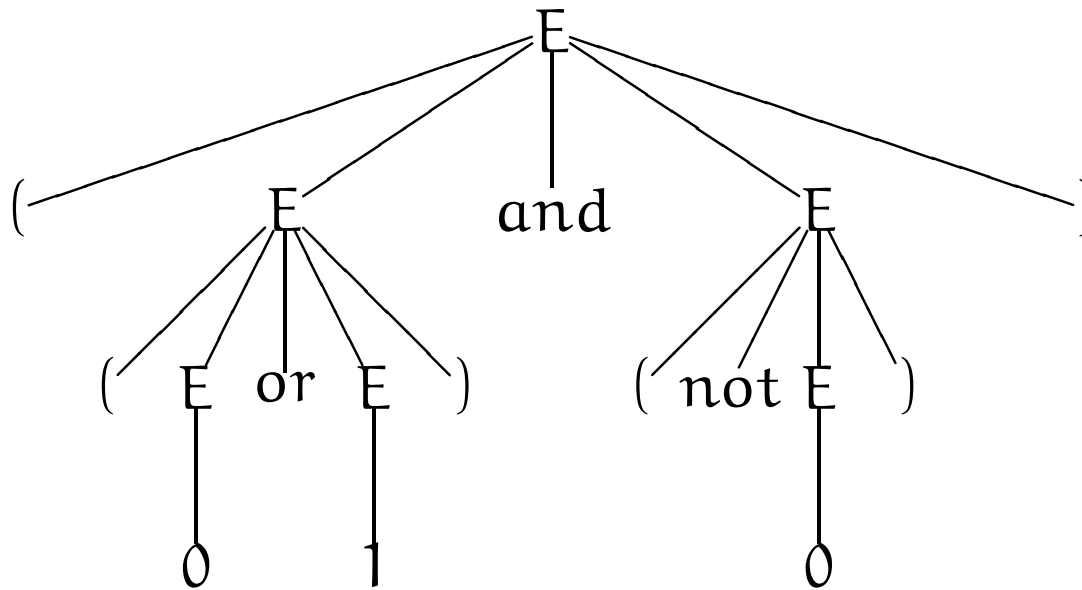
Example

$E \mapsto 01(E \text{ or } E) \mid (E \text{ and } E) \mid (\text{not } E) .$



Example

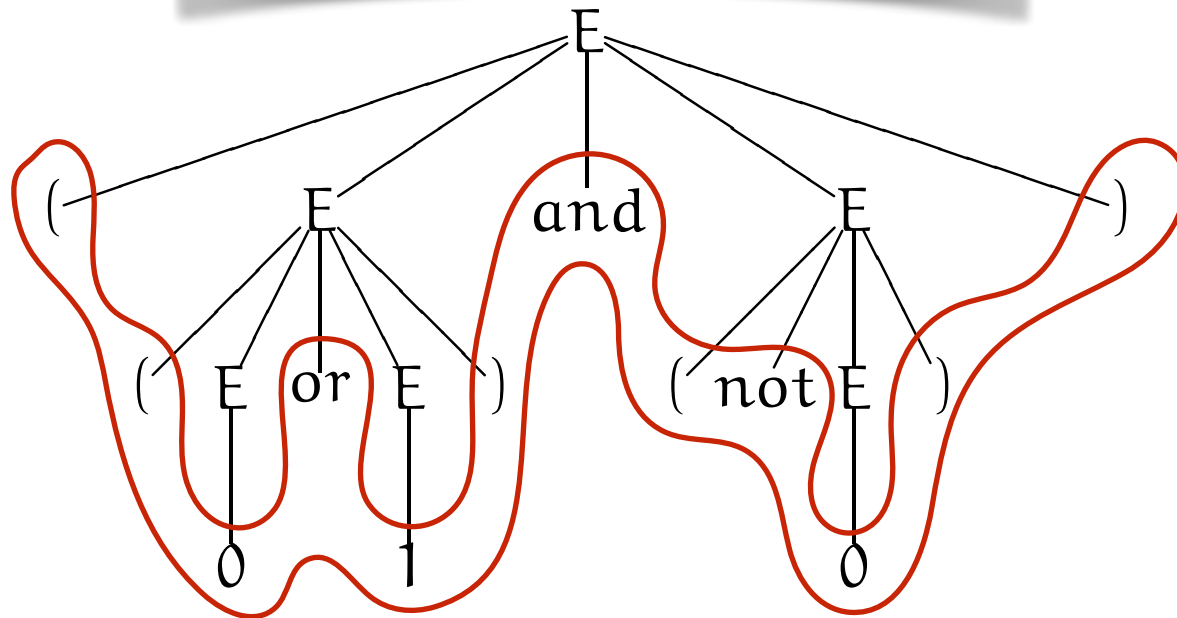
$E \mapsto 0|1|(E \text{ or } E)|(E \text{ and } E)|(\text{not } E) .$



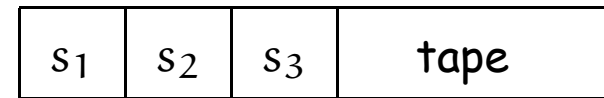
Example

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

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



Pushdown automata

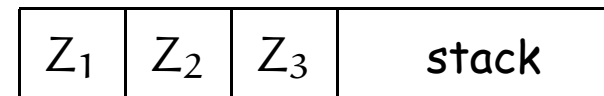


↑

q

Alphabet of stack symbols: R

↕



The stack head always scans the top symbol

It performs three basic operations:

Push: **add** a new symbols at the top of the stack

Pop: **read** and **remove** the top symbol

Empty: **verify** if the stack is empty

Pushdown automata

A push down automaton is $M = (Q, \Sigma, R, \delta, q_0, Z_0, F)$ where

- R is the alphabet of stack symbols,
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times R \rightarrow \mathcal{P}(Q \times R^*)$ is the transition function
- Z_0 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 is the current state
- w is the unread part of the input string (the remaining input)
- γ is the current contents of the stack

A move from one instantaneous description to another

will be denoted by

$$(q_0, aw, Zr) \mapsto (q_1, w, \gamma r) \text{ iff } (q_1, \gamma) \text{ belongs to } \delta(q_0, a, Z)$$

The language accepted by a pushdown automaton

Two ways to define the accepted language:

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

$$L_p(M) = \{x \in \Sigma^* : (q_0, x, Z_0) \mapsto_M^* (q, \varepsilon, \varepsilon), q \in Q\}$$

- with explicit final states F

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

Esempio

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

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

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

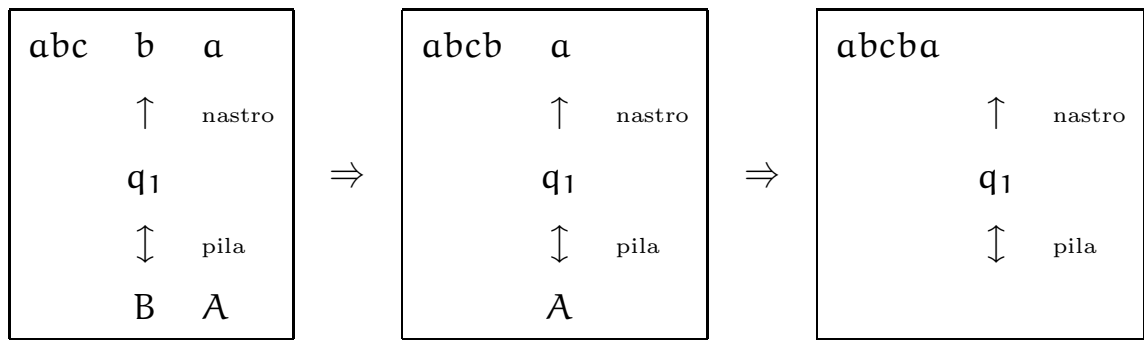
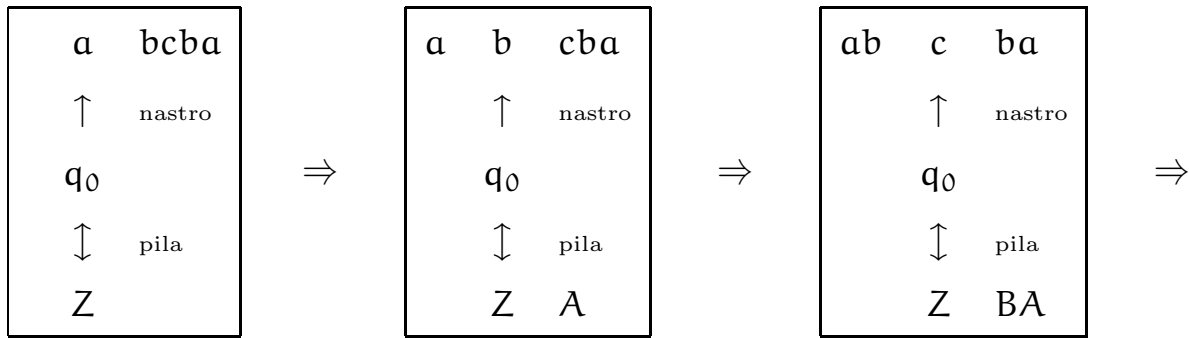
q_0	ε	a	b	c
Z		q_0, ZA	q_0, ZB	q_1, ε
A				
B				

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

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

Example: abcba

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



Example

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

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

q_0	ε	a	b
Z		q_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	q_1, ε		
A		q_1, ε	
B			q_1, ε

Exercises

Design a PDA to recognise the following languages:

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

$\{w \in \{0, 1\}^* \mid w \text{ has an equal number of 0's and 1's}\}$

Unfortunately...

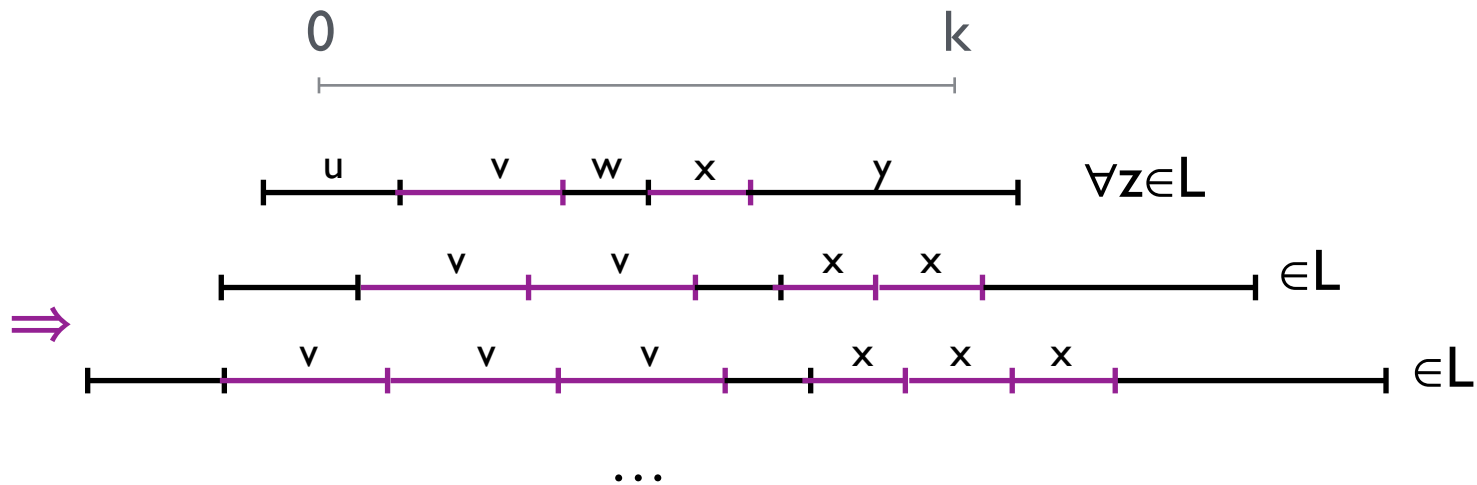
not all languages are Context Free !

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

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

L CF



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!**

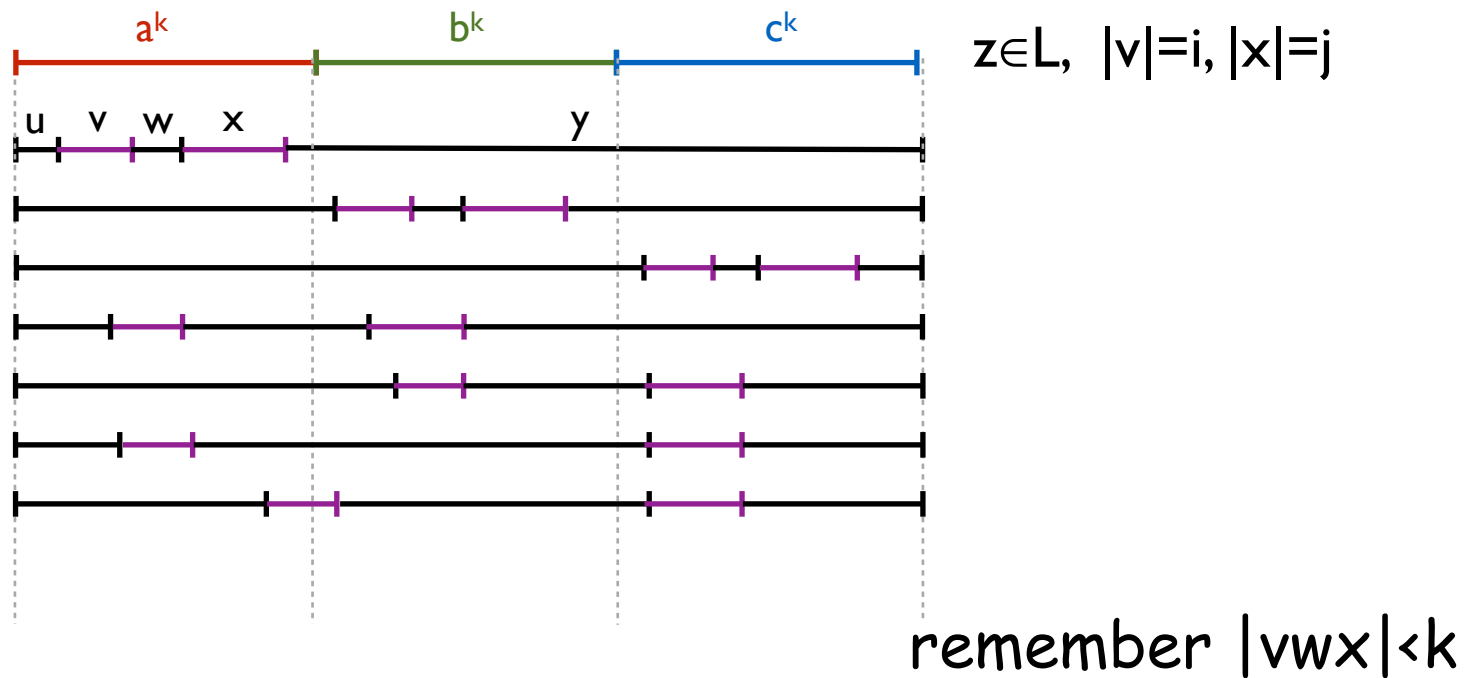
If $\forall k \in \mathbf{N} \exists z \in L. |z| \geq k$ for all possible splitting of the form

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

then **L is not context free!**

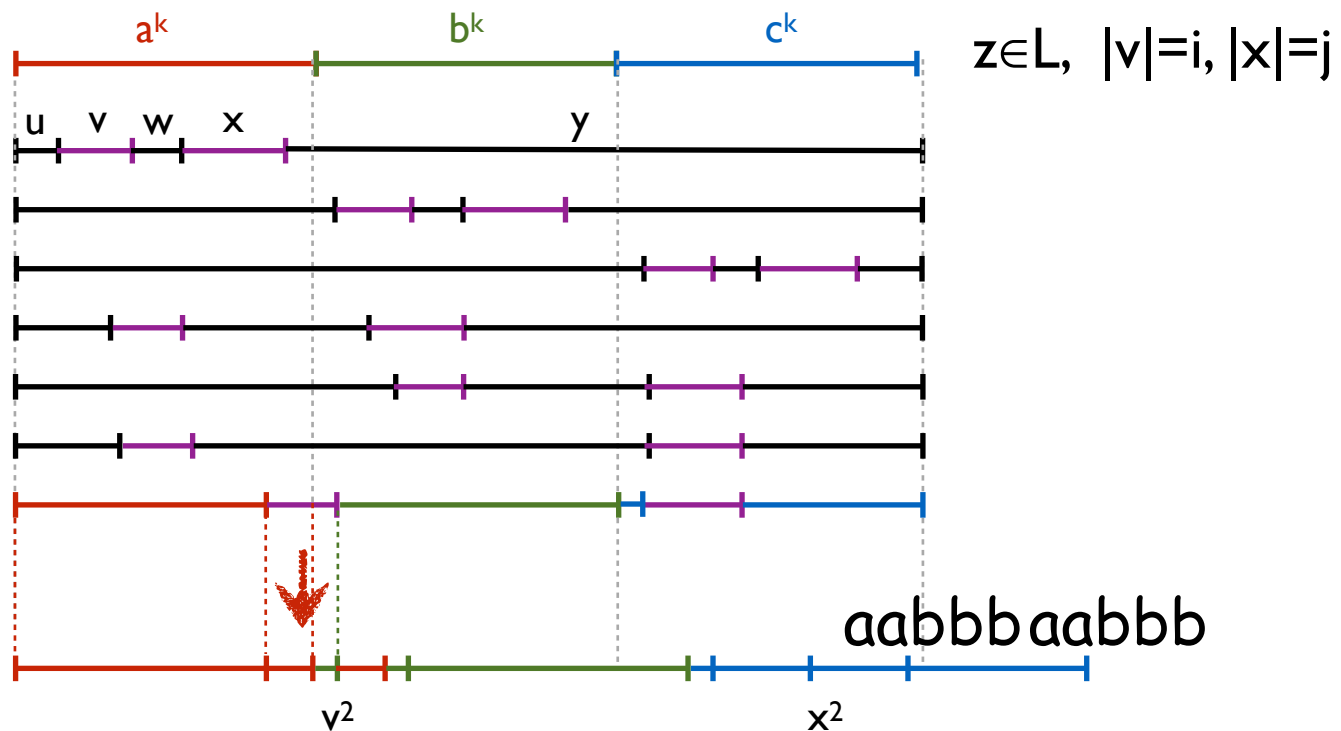
Example

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



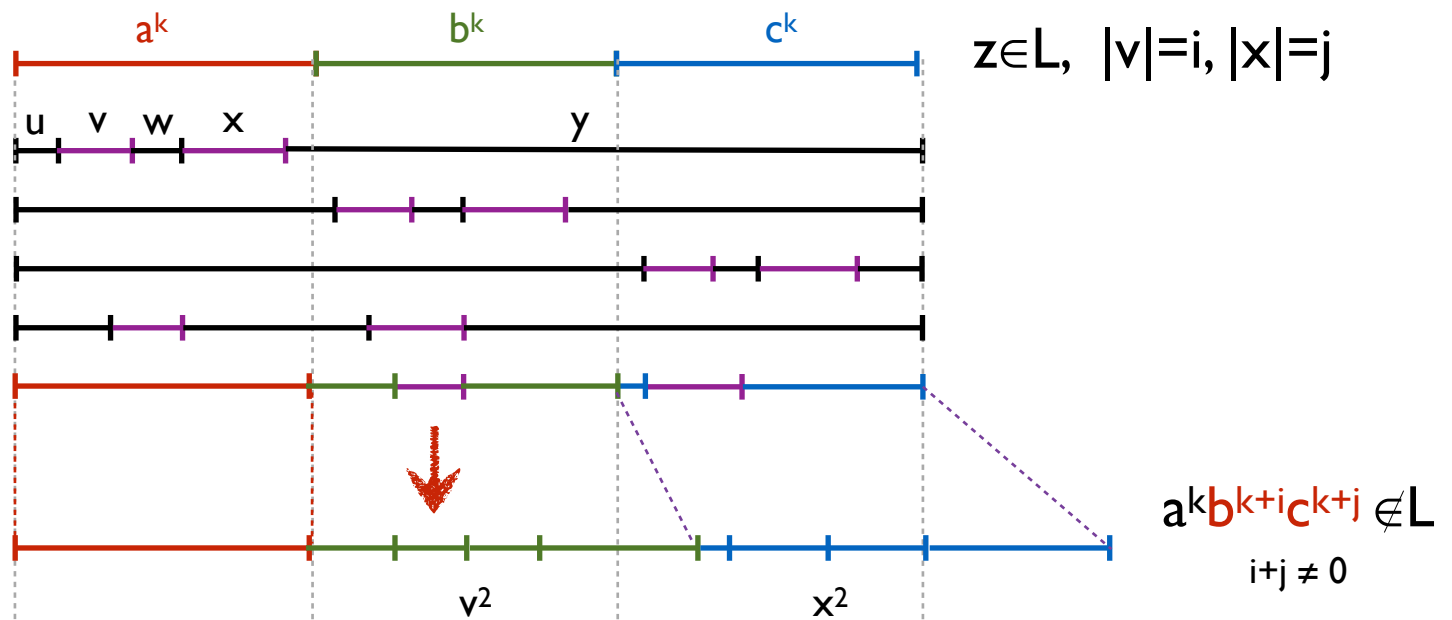
Example

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



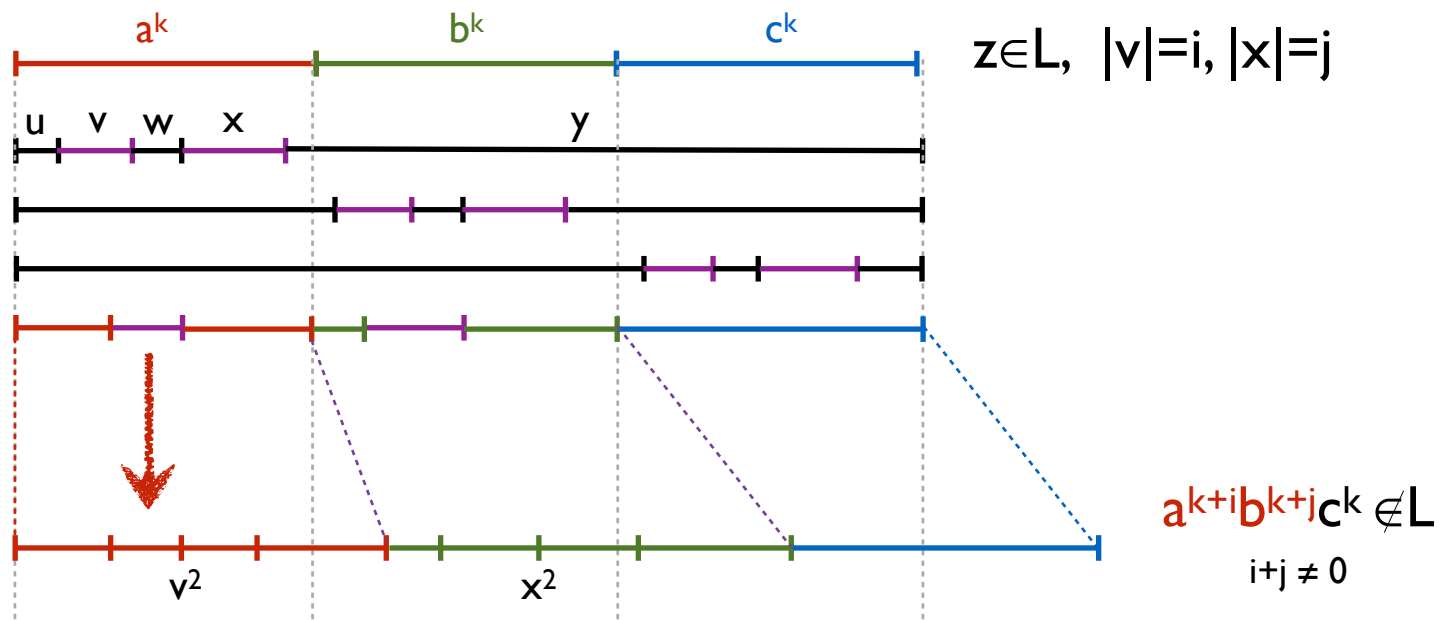
Example

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



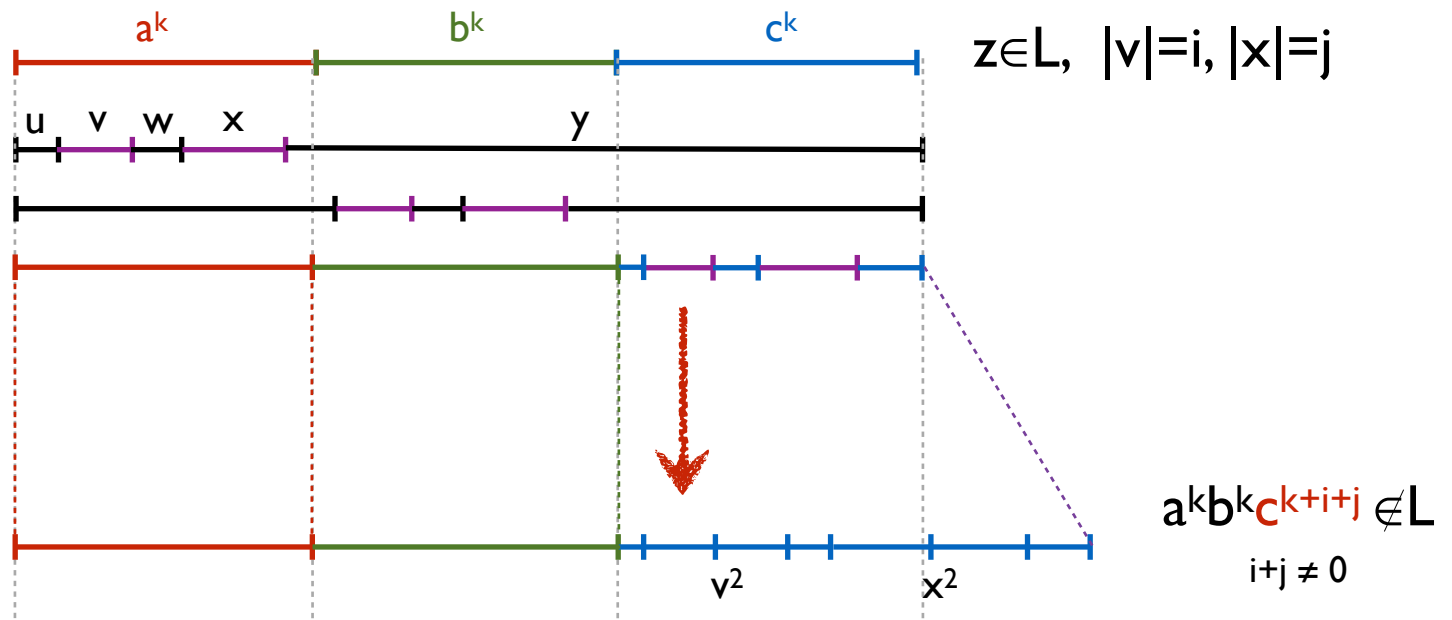
Example

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



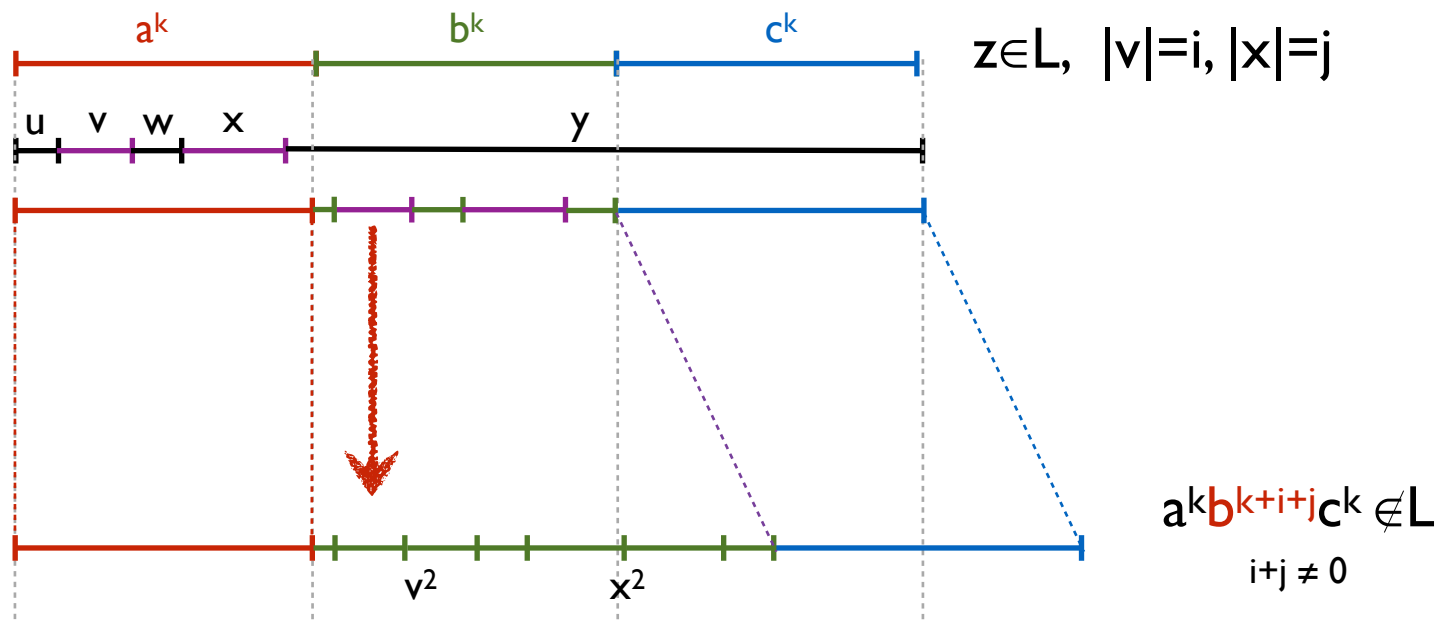
Example

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



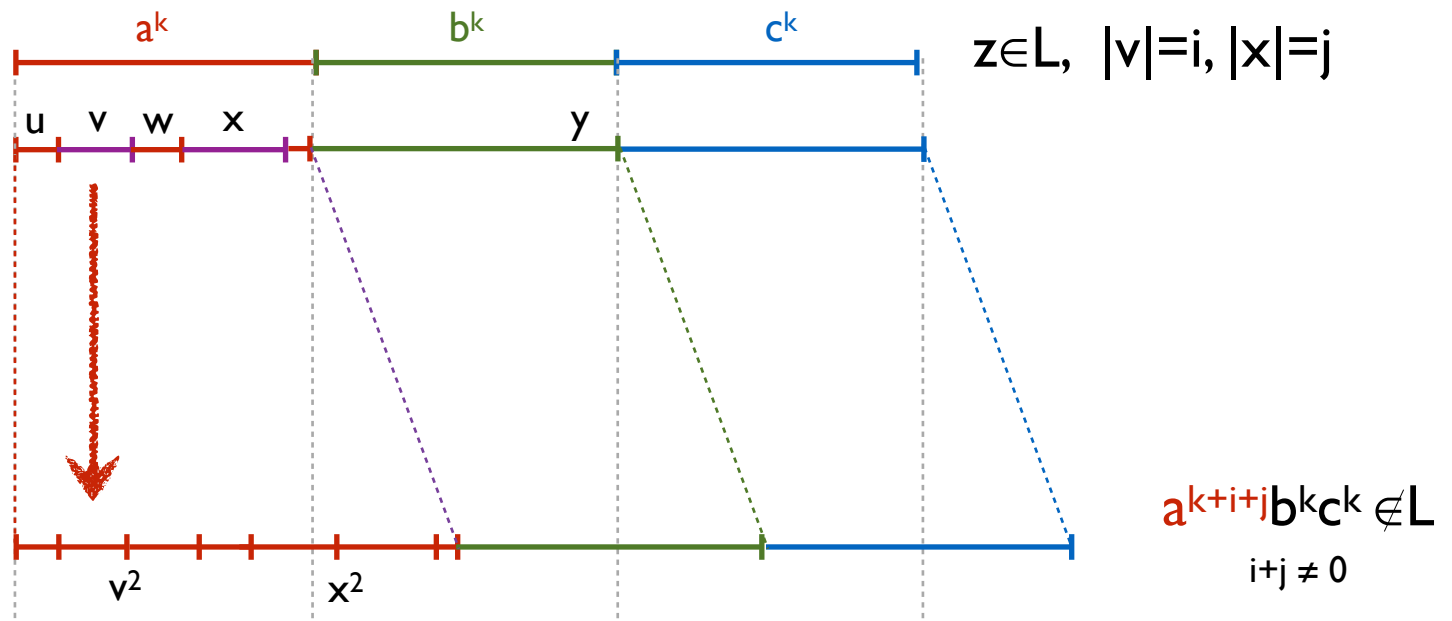
Example

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



Example

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



Exercises: are these languages context free?

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

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

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

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

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

$$L = \{w \in \{a, b\}^* \mid w \neq vv\}$$

If $|w|$ is odd then $w \in L$

$$A \rightarrow a \mid aAa \mid aAb \mid bAa \mid bAb$$

$$B \rightarrow b \mid aBa \mid aBb \mid bBa \mid bBb$$

$$S \rightarrow A \mid B \mid BA \mid AB \mid \epsilon$$

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 $U \rightarrow V$ such that $|U| \leq |V|$

Example

$S \rightarrow aSBC \mid aBC$

$bC \rightarrow bc$

$CB \rightarrow BC$

$cC \rightarrow cc$

$bB \rightarrow bb$

$aB \rightarrow ab$

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

Complexity of Languages Problems

	Regular Grammar Type 3	Context Free Grammar Type 2	Context Sensitive Grammar Type 1	Unrestricted Grammar Type 0
Is $W \subseteq L(G)$?	P	P	PSPACE	U
Is $L(G)$ empty?	P	P	U	U
Is $L(G_1) = L(G_2)$?	PSPACE	U	U	U

Examples of Language Hierarchy

The expressing 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 = $\{a^n b^n \mid n \in \mathbf{N}\}$ is context-free, but not regular

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

Relationships between Languages and Automata

A language is :

regular

context-free

context-sensitive

phrase-structure

iff accepted by

finite-state automata

pushdown automata

linear-bounded automata

Turing machine

Chomsky's Hierarchy

