

Operations on languages: recap.

Union: $A \cup B$

Intersection: $A \cap B$

Difference: $A \setminus B$

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

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.

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

- \emptyset denoting the **empty set**,
- ε denoting the set $\{\varepsilon\}$,
- 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 $R \cup S$, RS and R^* , respectively.

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

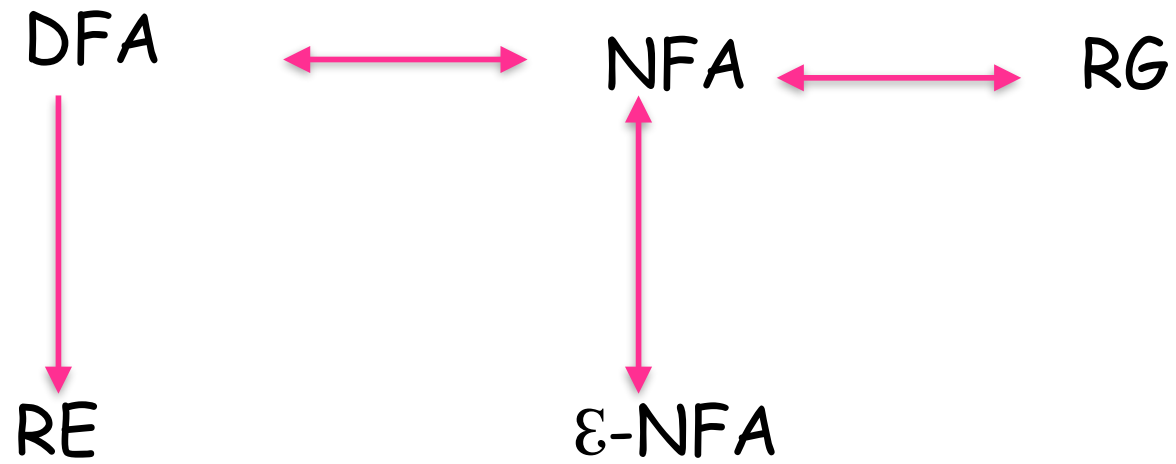
Examples

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

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

- $a|b^*$ denotes
 $\{\varepsilon, "a", "b", "bb", "bbb", \dots\}$
- $(a|b)^*$ denotes
all the strings formed with "a" and "b"
- $ab^*(c|\varepsilon)$ 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 , then there is a regular expression r such that $L(D)=L(r)$.

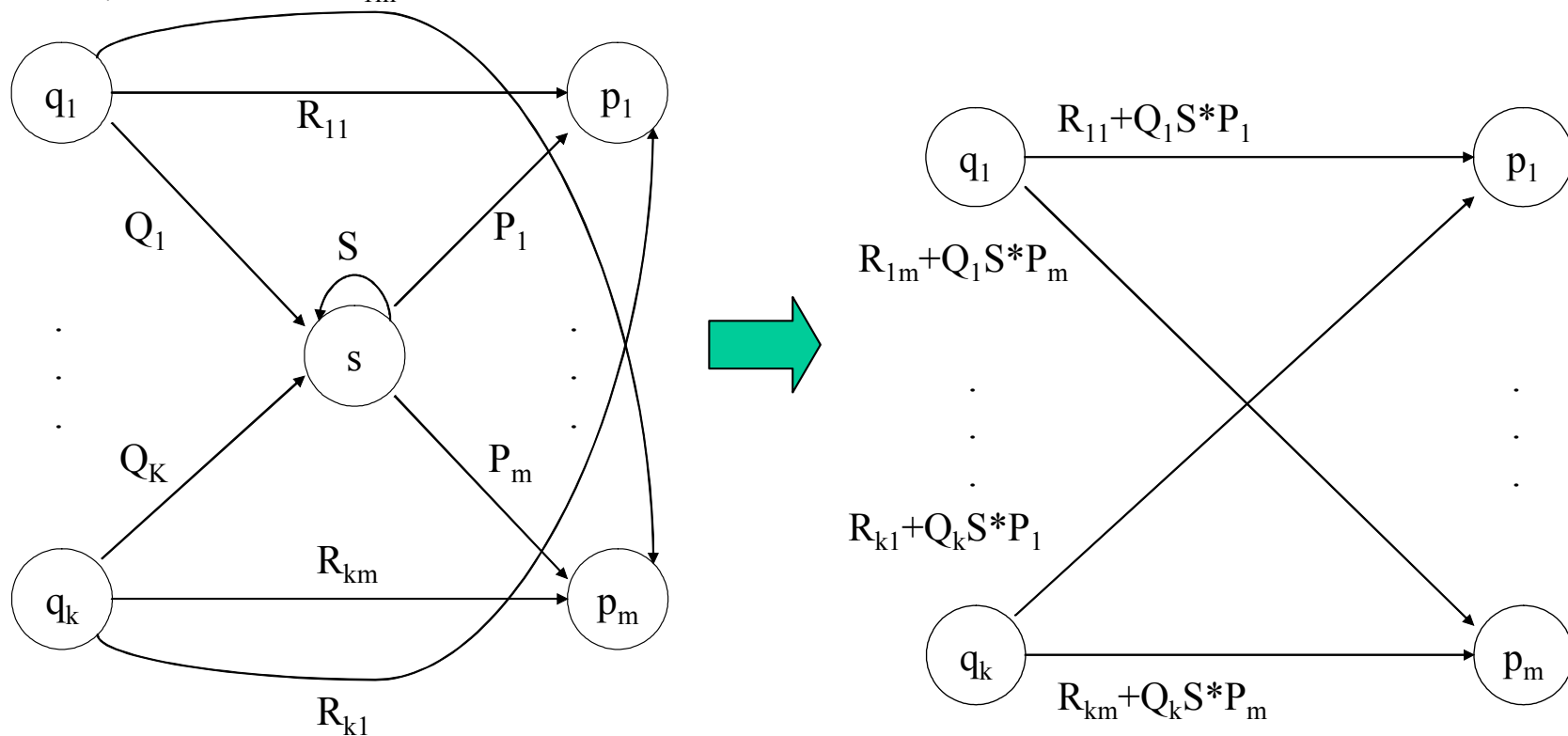
Construction:

- 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 .



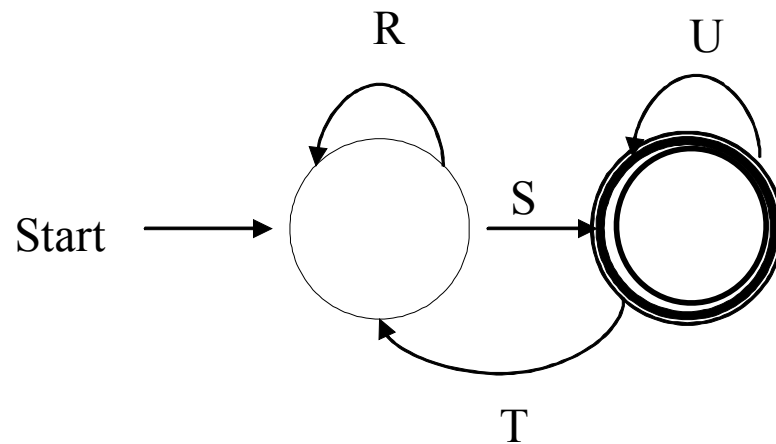
DFA to RE via State Elimination

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)

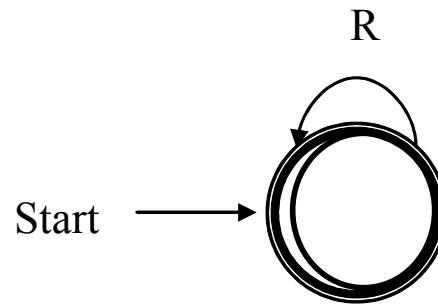
If the two states are different, we will have an automaton that looks like the following:



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:



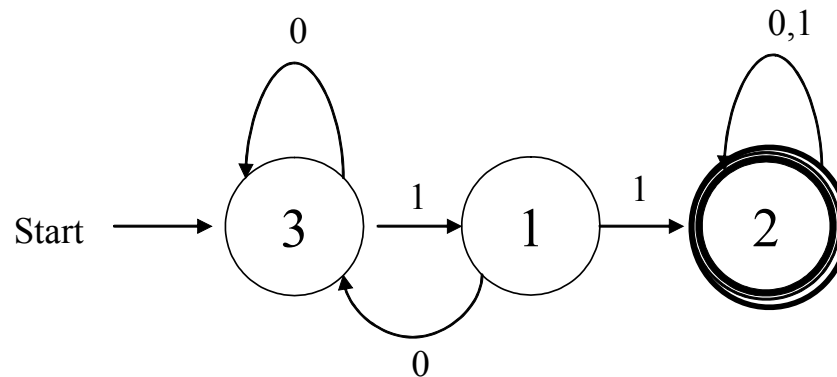
We can describe this automaton as simply R^* .

DFA to RE State Elimination (4)

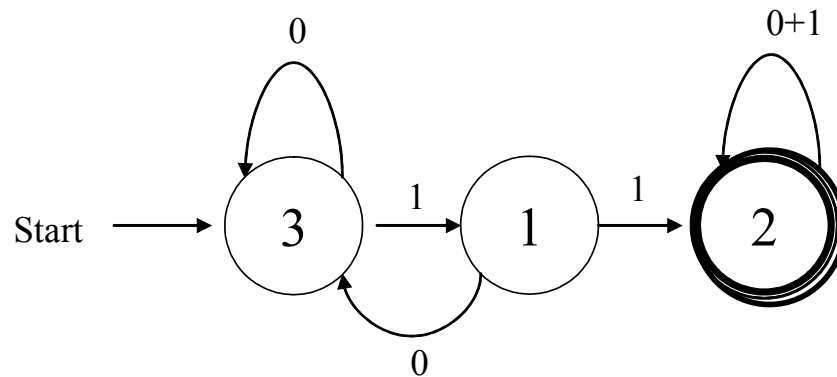
If there are n accepting states, we must repeat the above steps for each accepting states to get n different regular expressions, $R_1, R_2, \dots R_n$. For each repeat we turn any other accepting state to non-accepting. The desired regular expression for the automaton is then the union of each of the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

DFA \rightarrow RE Example

- Convert the following to a RE

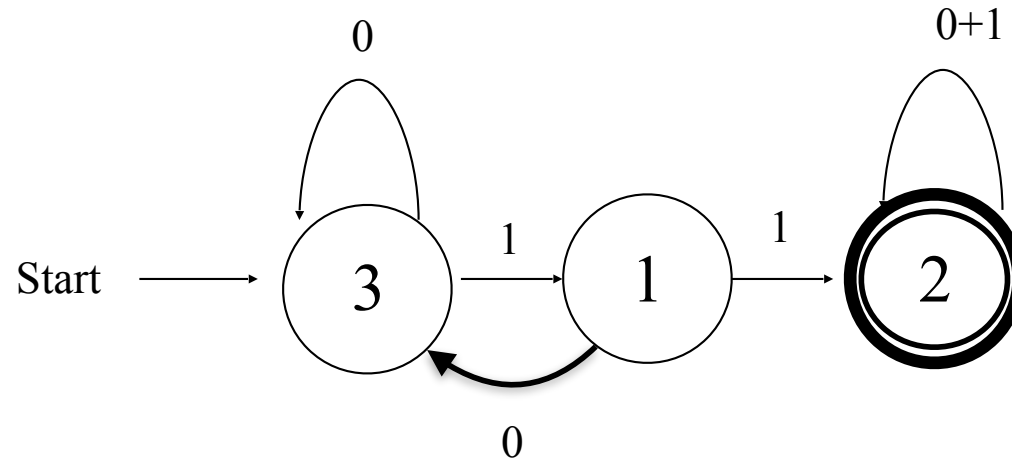


- First convert the edges to RE's:



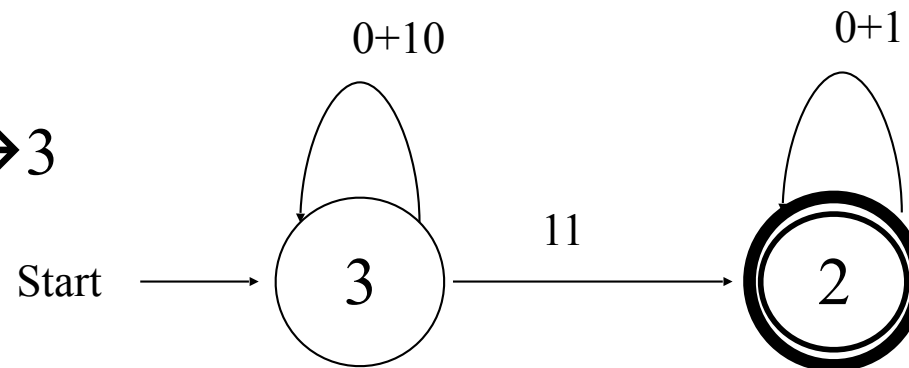
DFA \rightarrow RE Example (2)

- we want to eliminate State 1:



- obtaining:

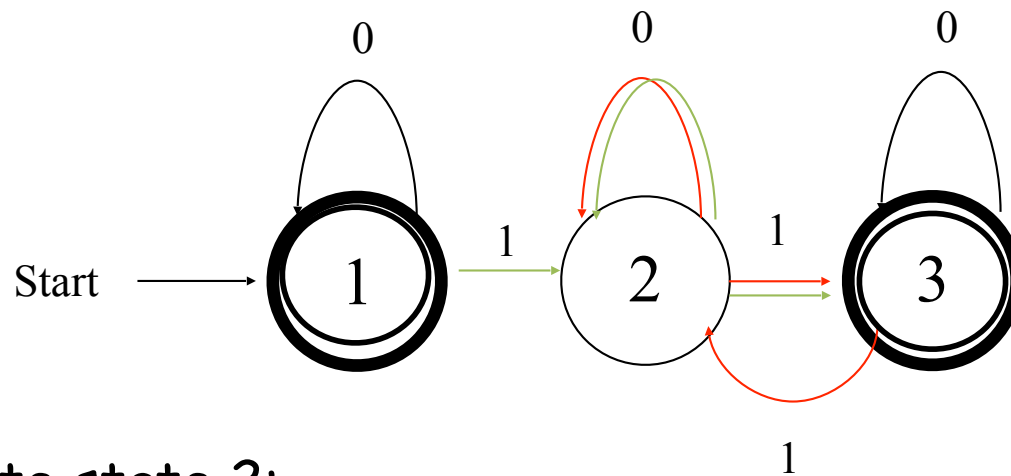
Note edge from $3 \rightarrow 3$



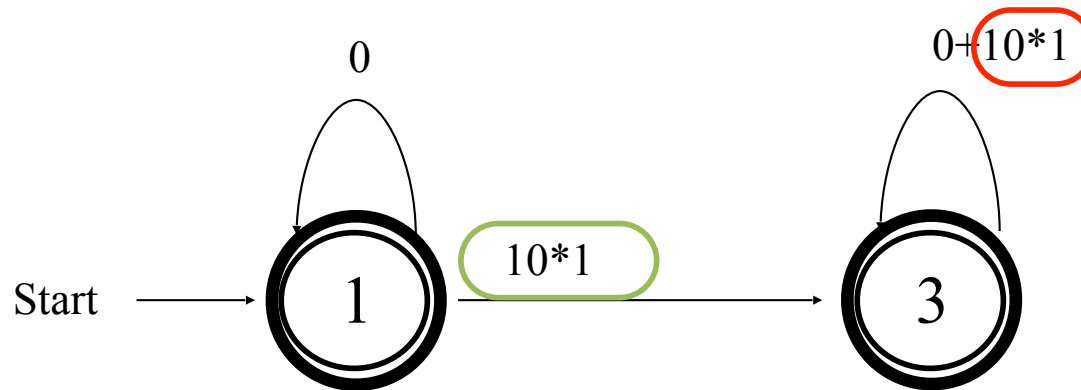
Answer: $(0+10)^*11(0+1)^*$

Third Example

- Automata that accepts even number of 1's

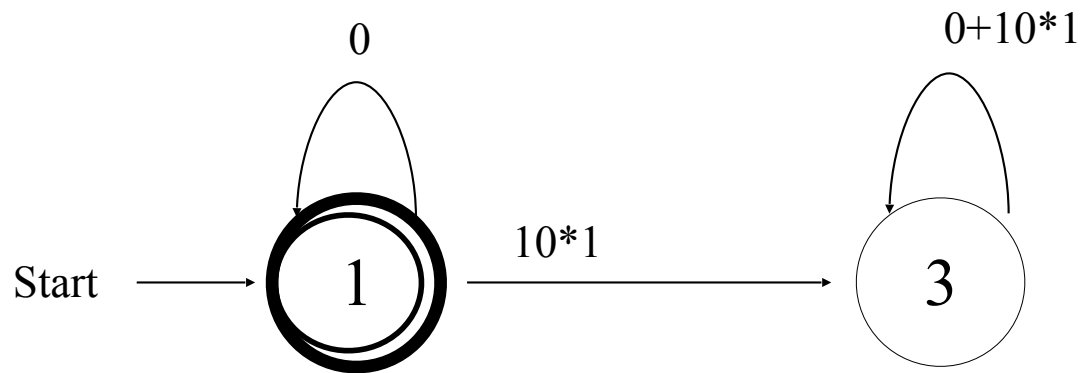
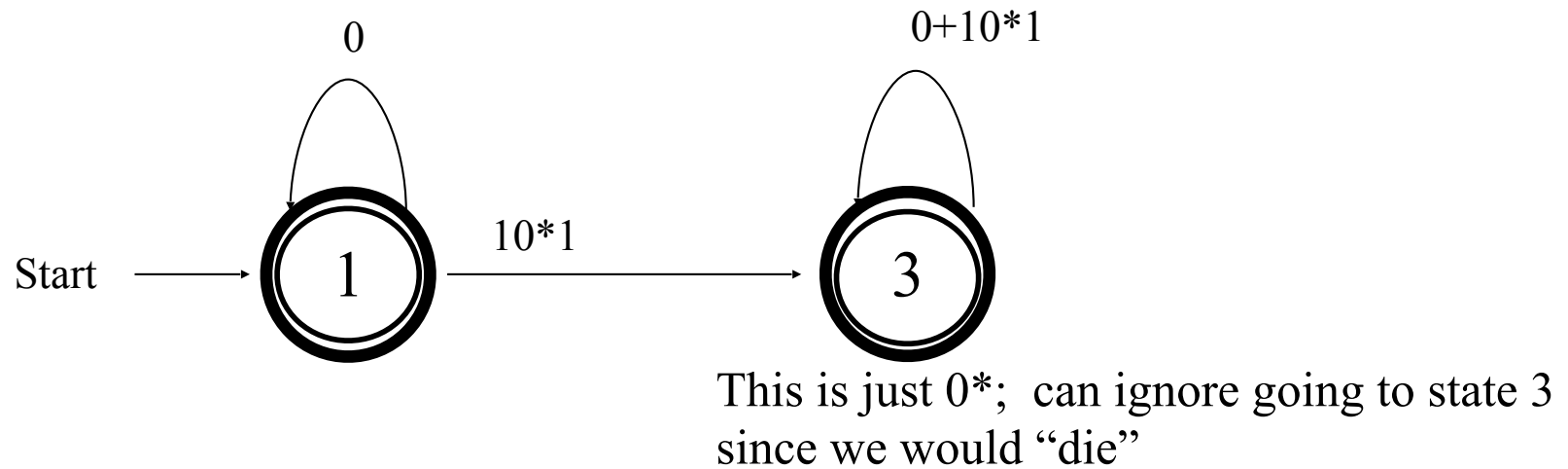


- Eliminate state 2:



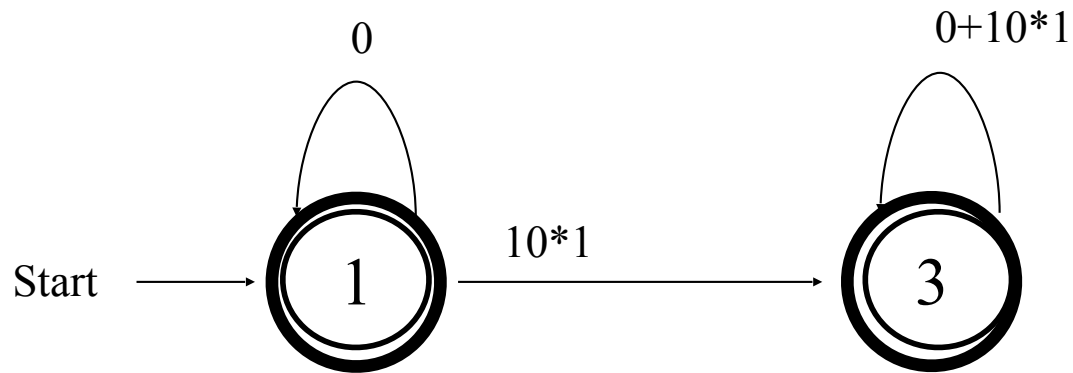
Third Example (2)

- Two accepting states, turn off state 3 first

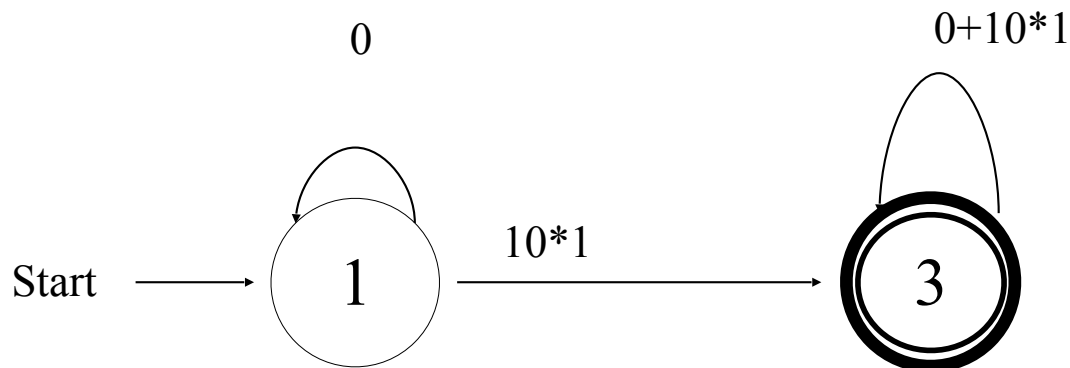


Second Example (3)

- Turn off state 1 second:

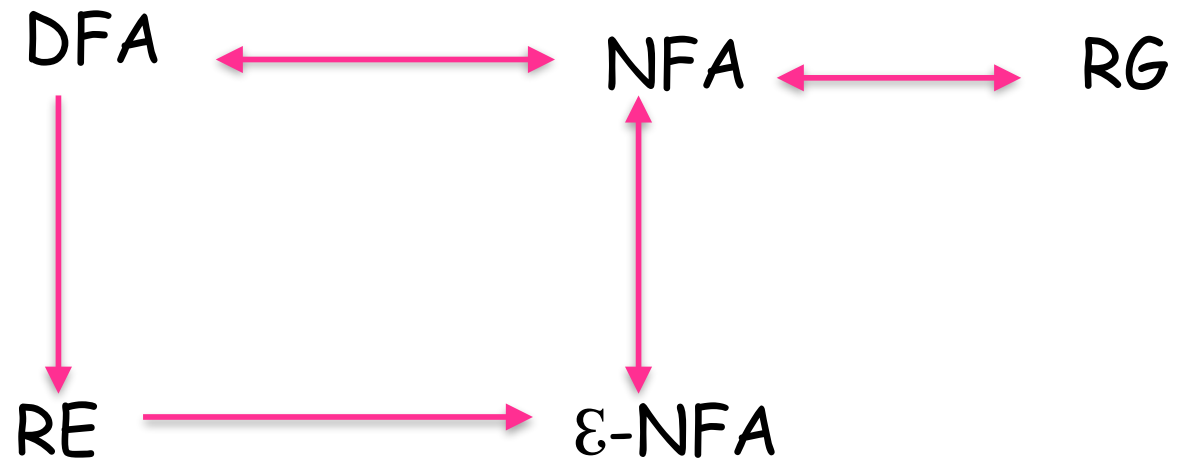


This is just $0^*10^*1(0+10^*1)^*$



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

Roadmap

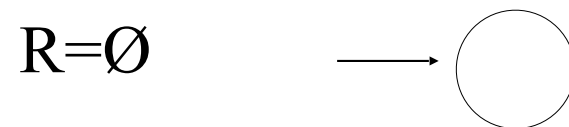
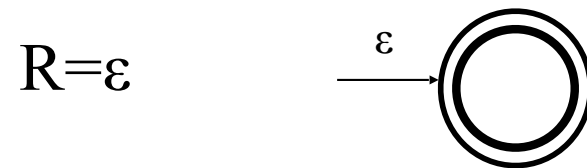
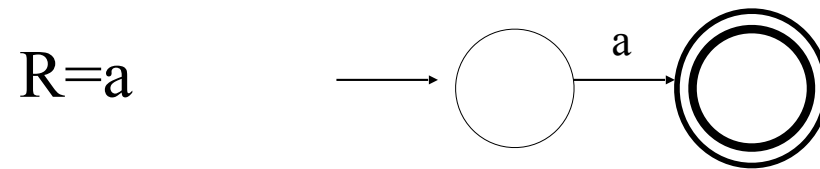


Converting a RE to an Automata

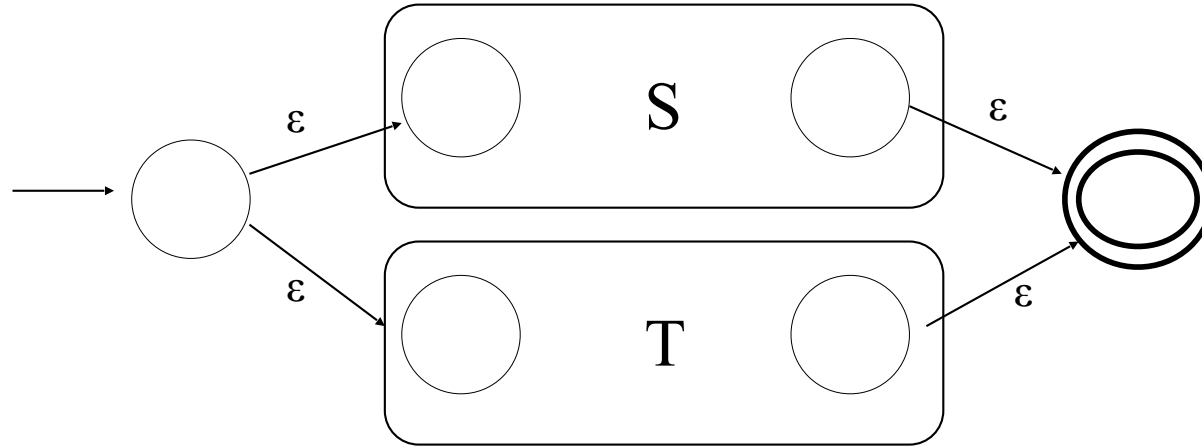
- We can convert a RE to an ε -NFA
 - Inductive construction
 - Start with a simple basis, use that to build more complex parts of the NFA

RE to ϵ -NFA

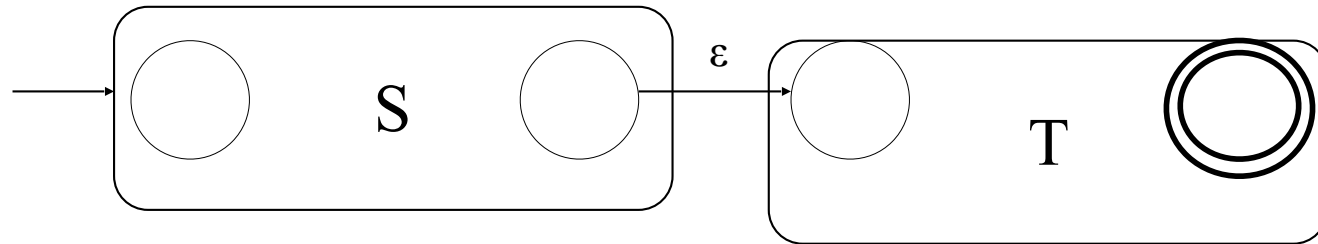
- Basis:



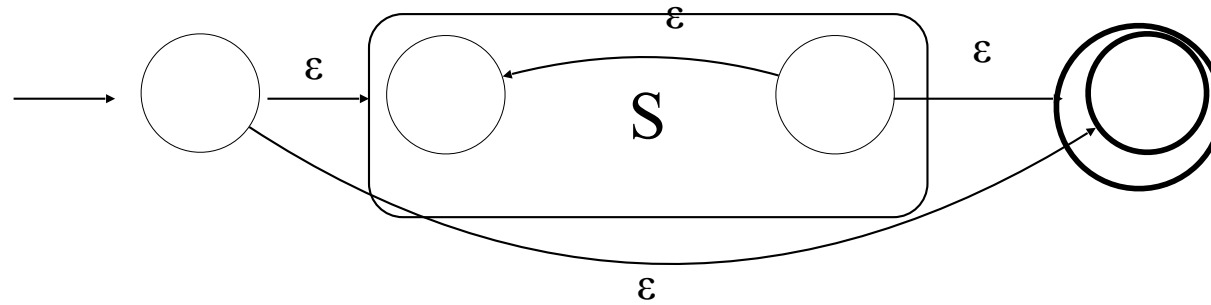
$R=S+T$



$R=ST$

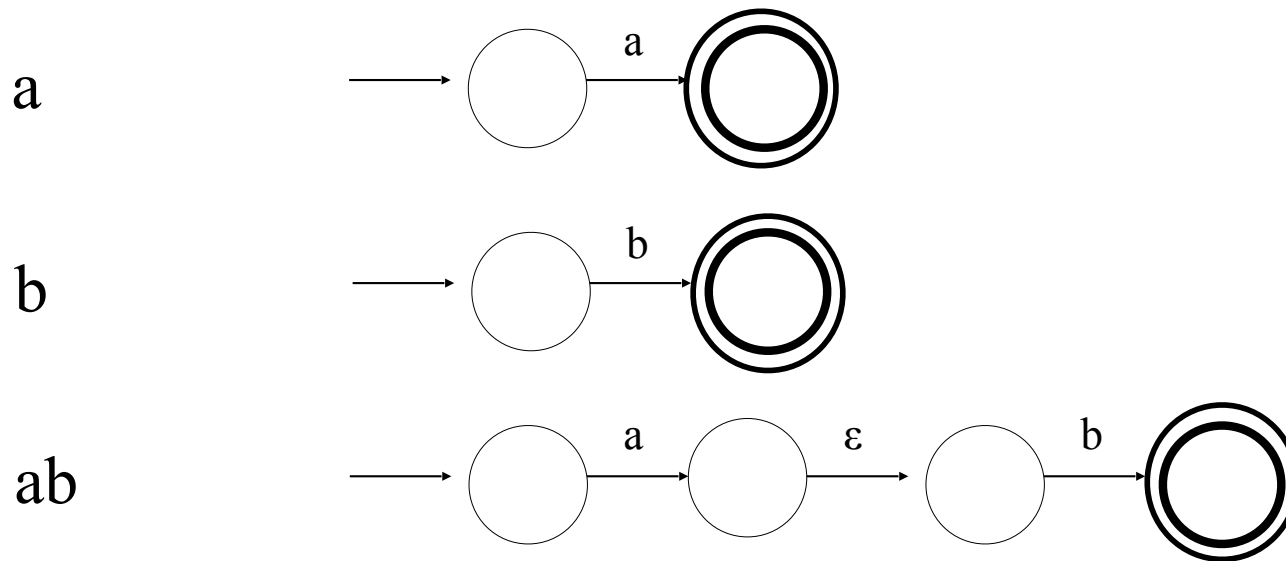


$R=S^*$



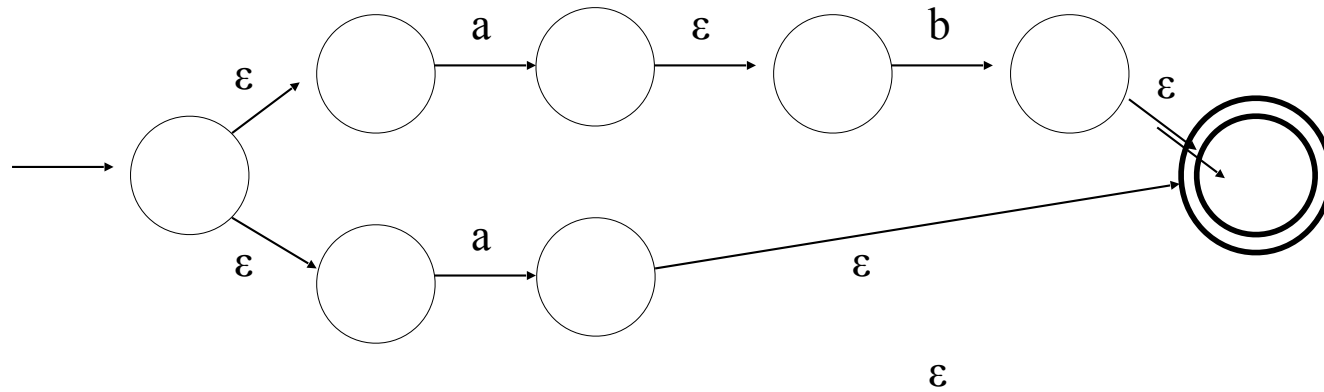
RE to ϵ -NFA Example

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

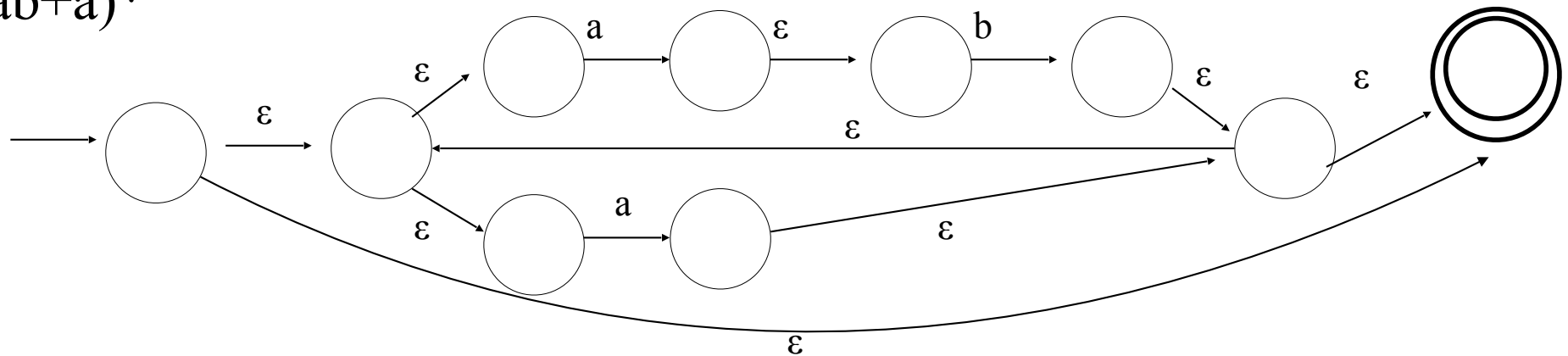


RE to ϵ -NFA Example (2)

$ab+a$



$(ab+a)^*$

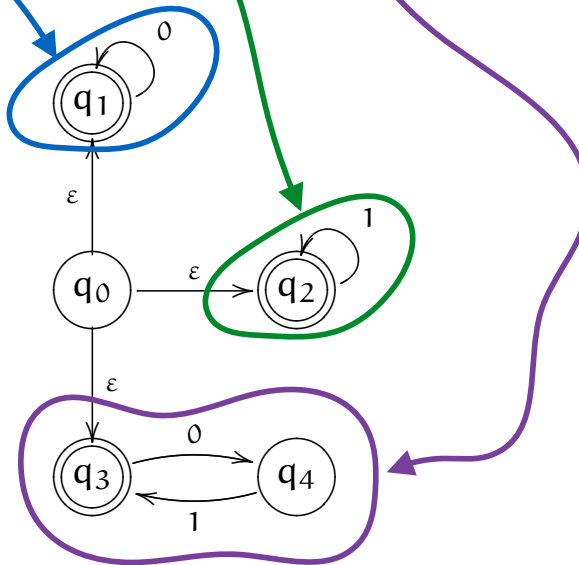


Esempio: from RE to ϵ -NFA

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

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

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

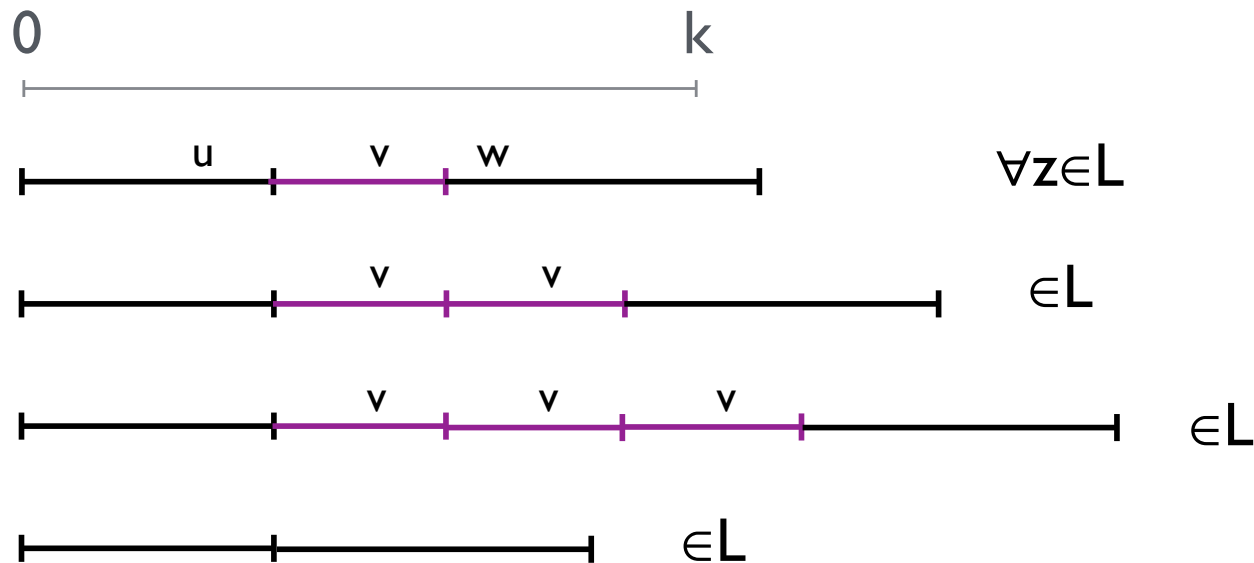
Not all languages are regular!

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

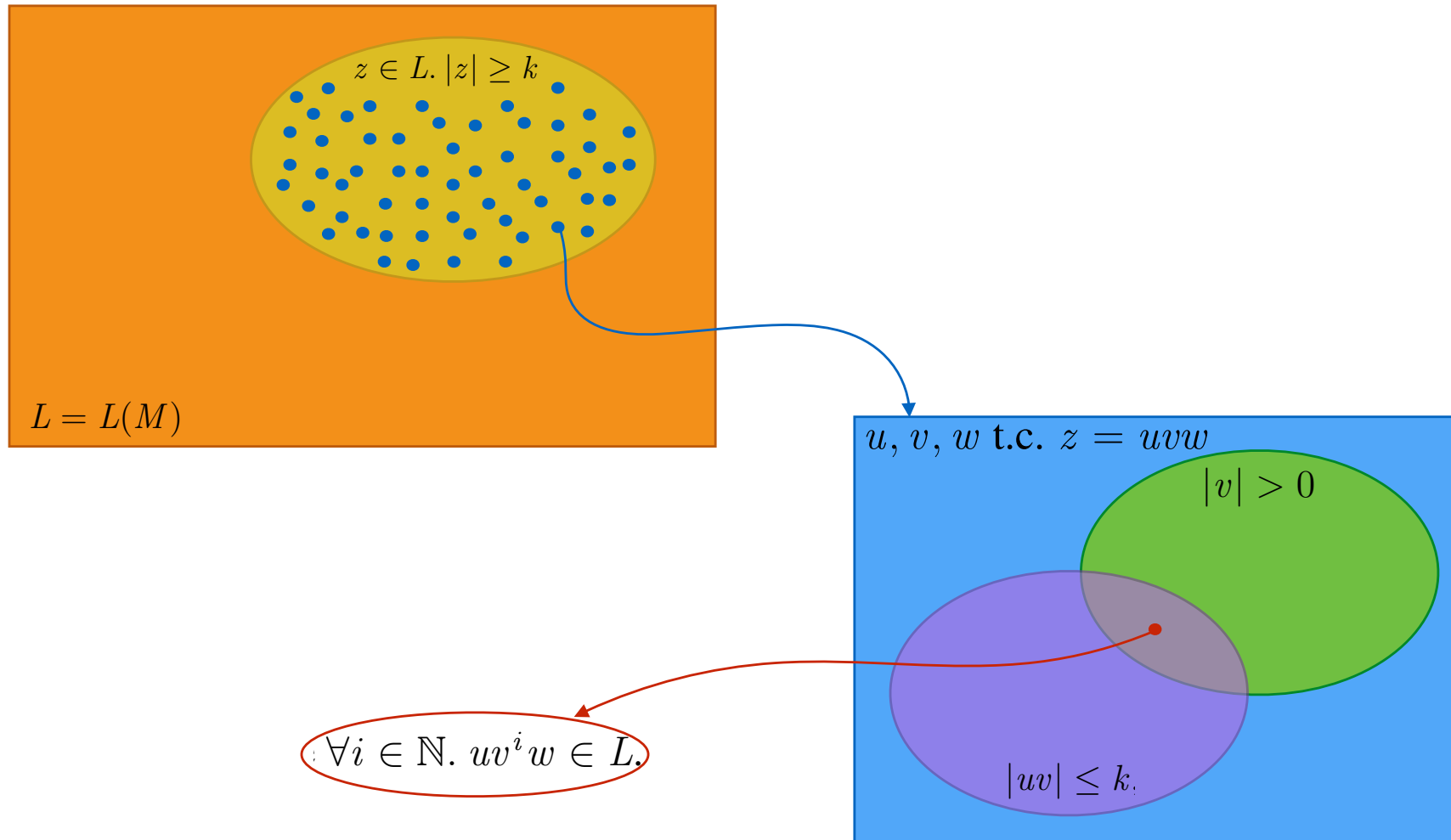
Pumping Lemma

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

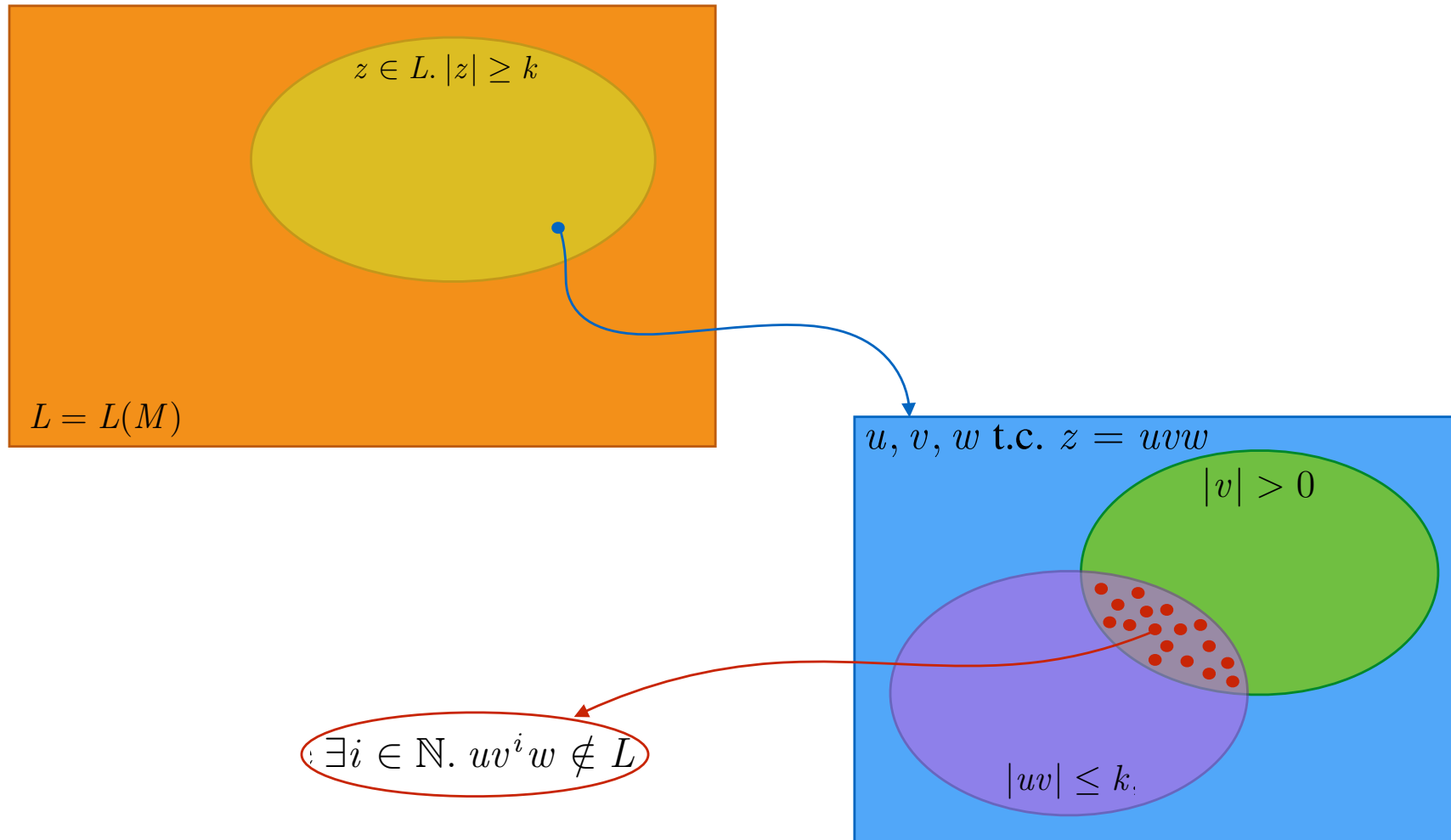
$z = uvw$ with $|uv| \leq k, |v| > 0$ such that $\forall i \in \mathbf{N}, uv^i w \in L$



Meaning of the PL



Negating the PL



Negating the PL

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

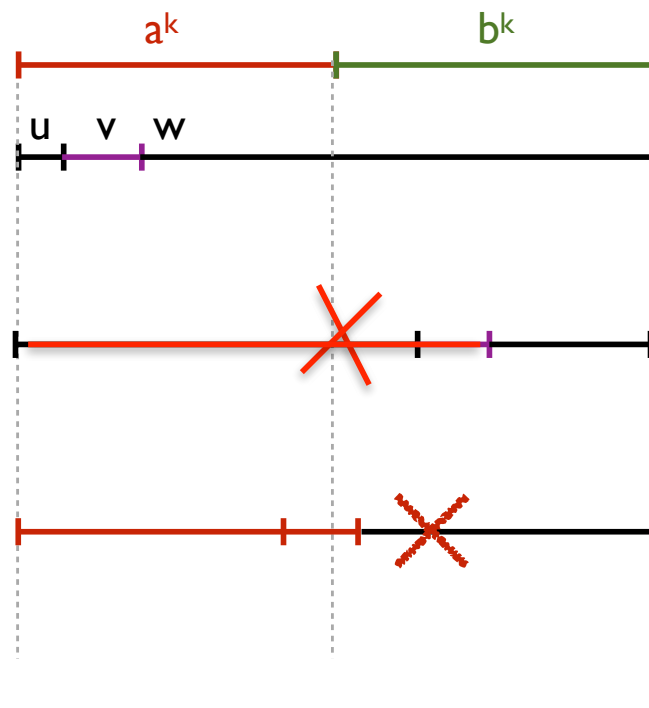
If $\forall k \in \mathbf{N} \exists z \in L. |z| \geq k$ for all possible splitting

$z = uvw$ 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

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

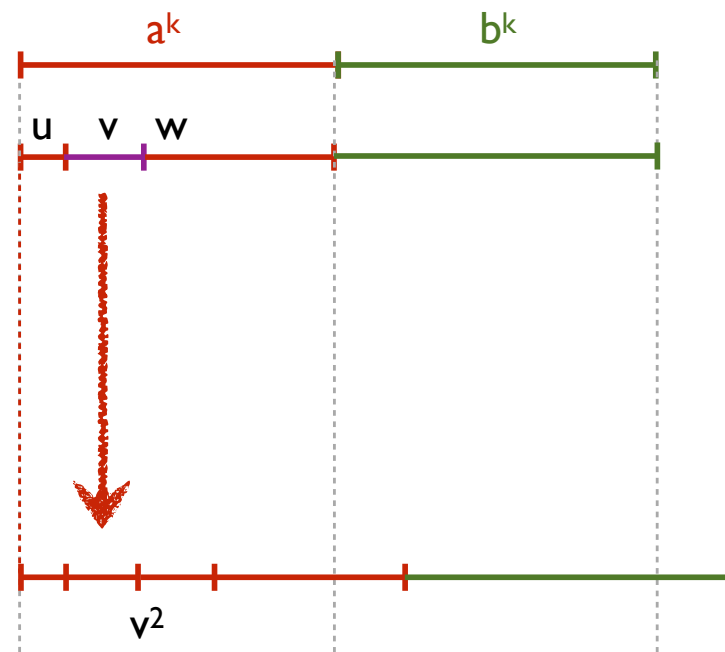


$z \in L$, $|v| = i$,

$|uv| > k$

Esempio

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

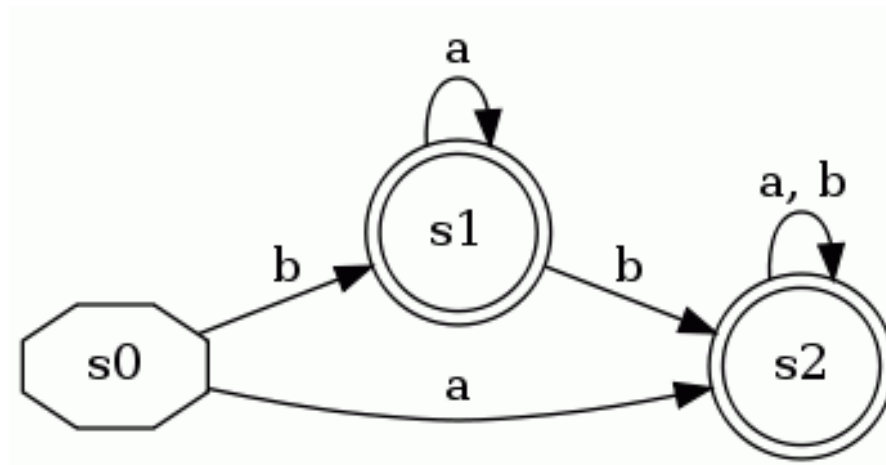
Decision Properties:

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*, then, is to automatically transform a given DFA into a state-minimized DFA
 - Several algorithms and variants are known

DFA Minimization Algorithm

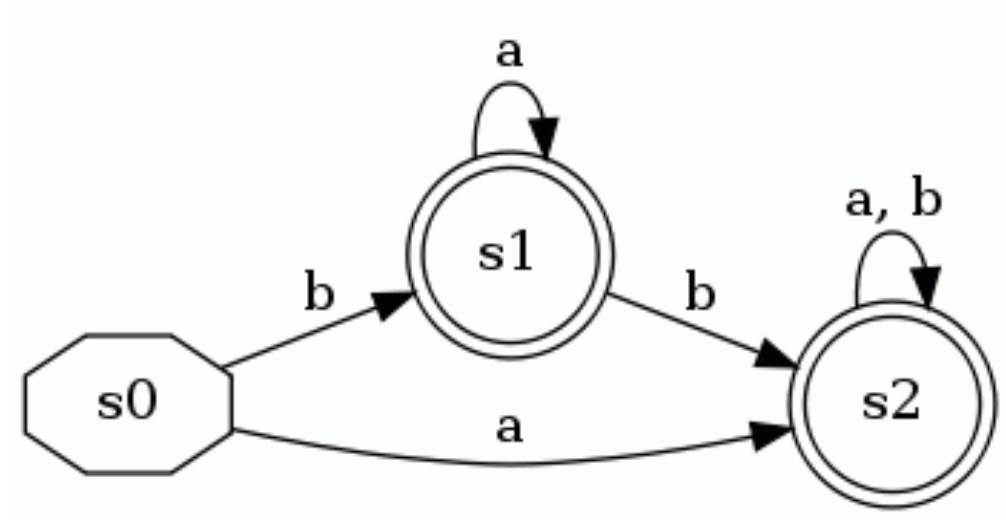
- Recall that a DFA $M=(Q, \Sigma, \delta, q_0, F)$
- Two states p and q are distinct if
 - p in F and q not in F or vice versa, or
 - for some a in Σ , $\delta(p, a)$ and $\delta(q, a)$ are distinct
- Using this inductive definition, we can calculate which states are distinct

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
 - $\text{DISTINCT}(p,q) = \varepsilon$
- 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

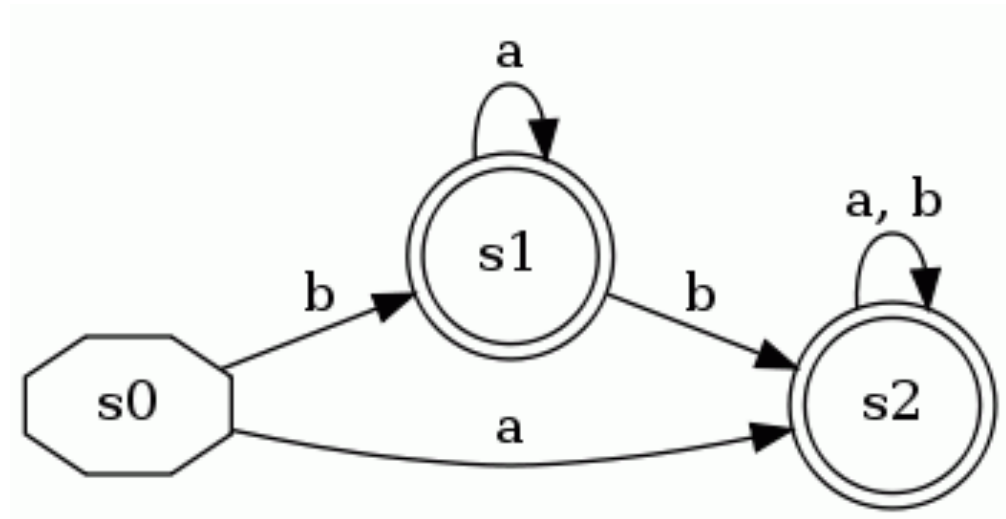
Very Simple Example

s0			
s1			
s2			
	s0	s1	s2



Very Simple Example

s0			
s1	ϵ		
s2	ϵ		
	s0	s1	s2



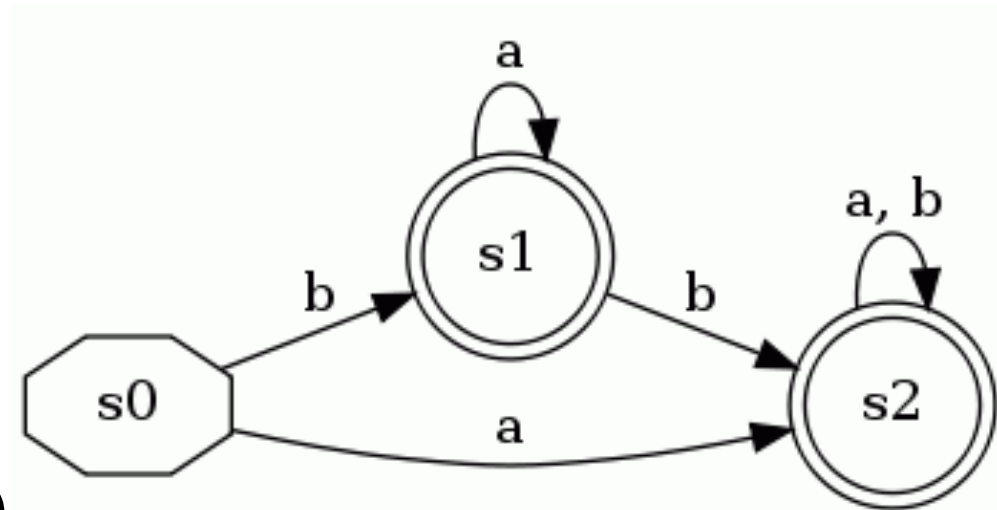
Label pairs with ϵ where one is a final state and the other is not

Very Simple Example

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

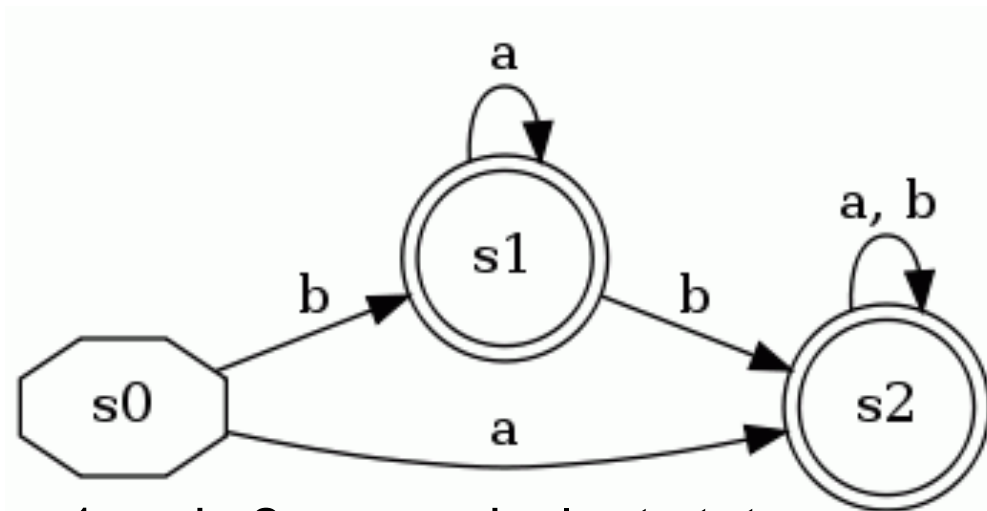
s0			•
s1	ϵ		
s2	ϵ		
	s0	s1	s2

Main loop (no changes occur)



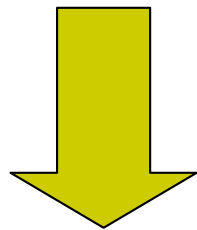
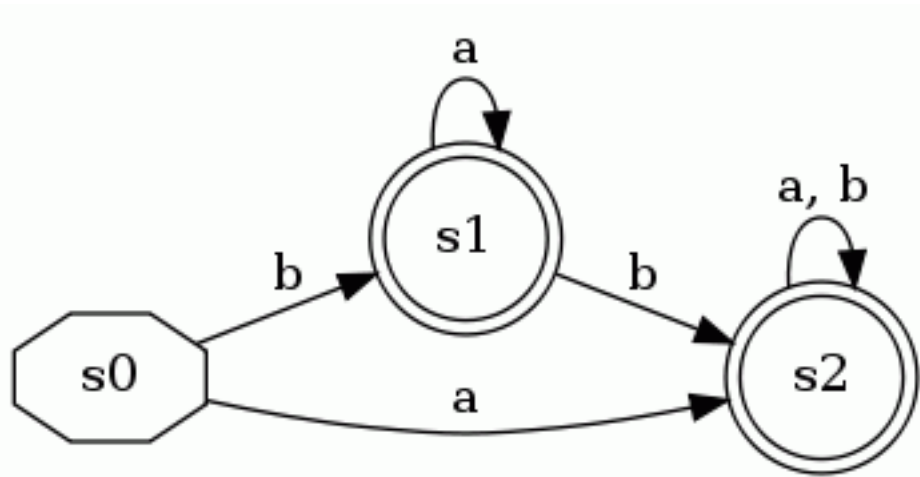
Very Simple Example

s0			
s1	ϵ		
s2	ϵ		
	s0	s1	s2

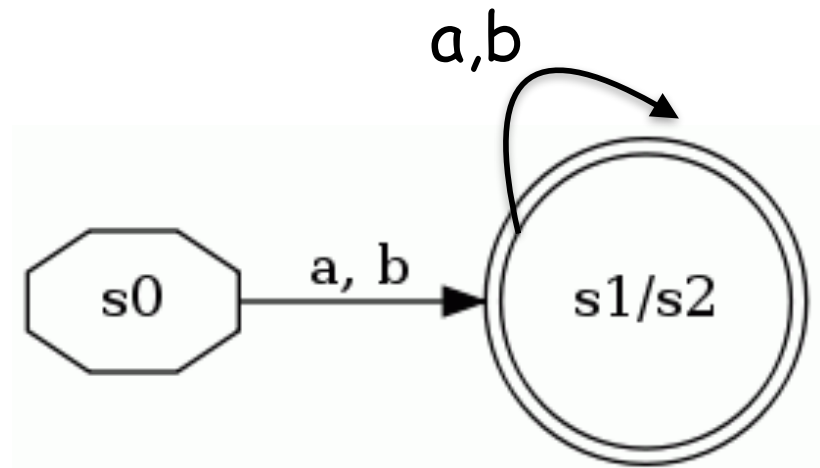
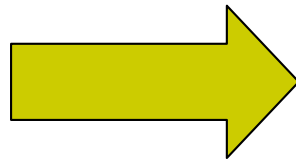


$\text{DISTINCT}(s1, s2)$ is empty, so s1 and s2 are equivalent states

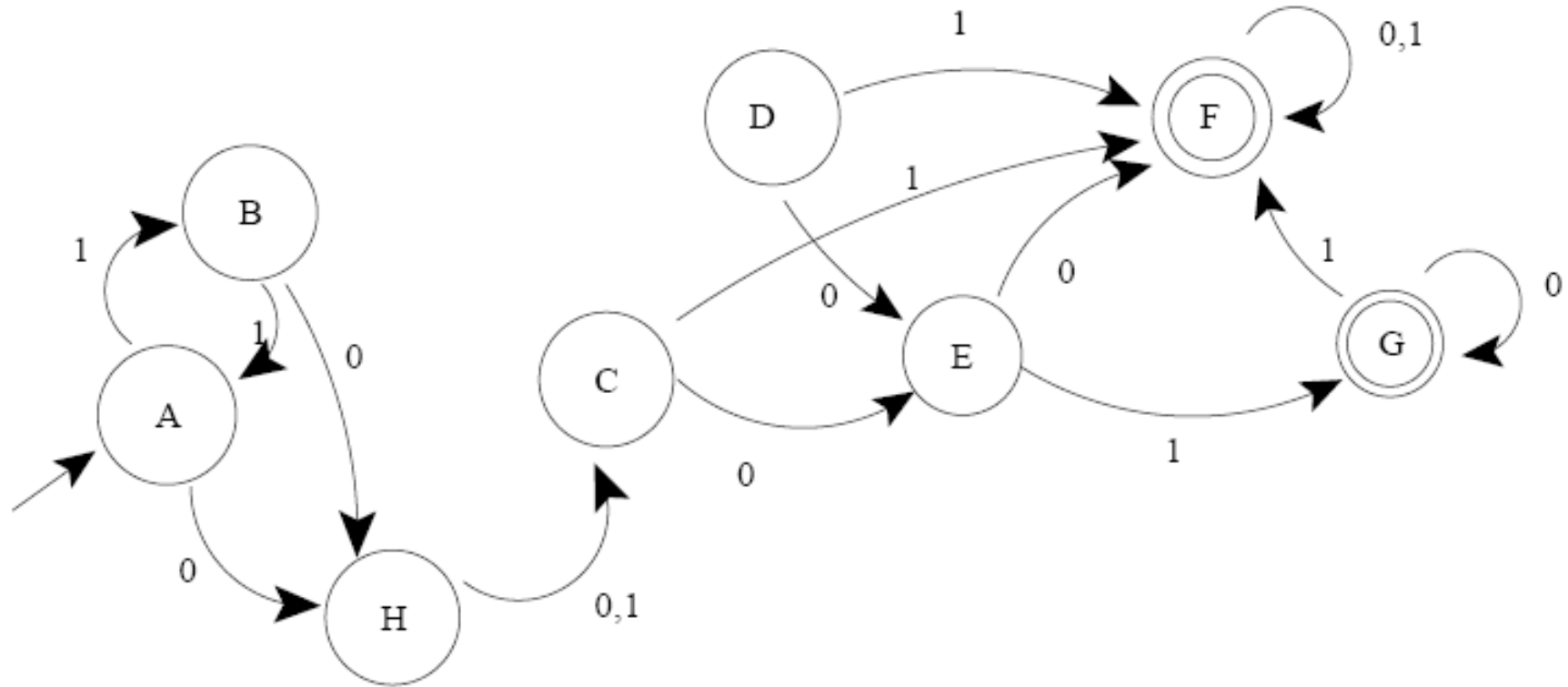
Very Simple Example



Merge s_1 and s_2

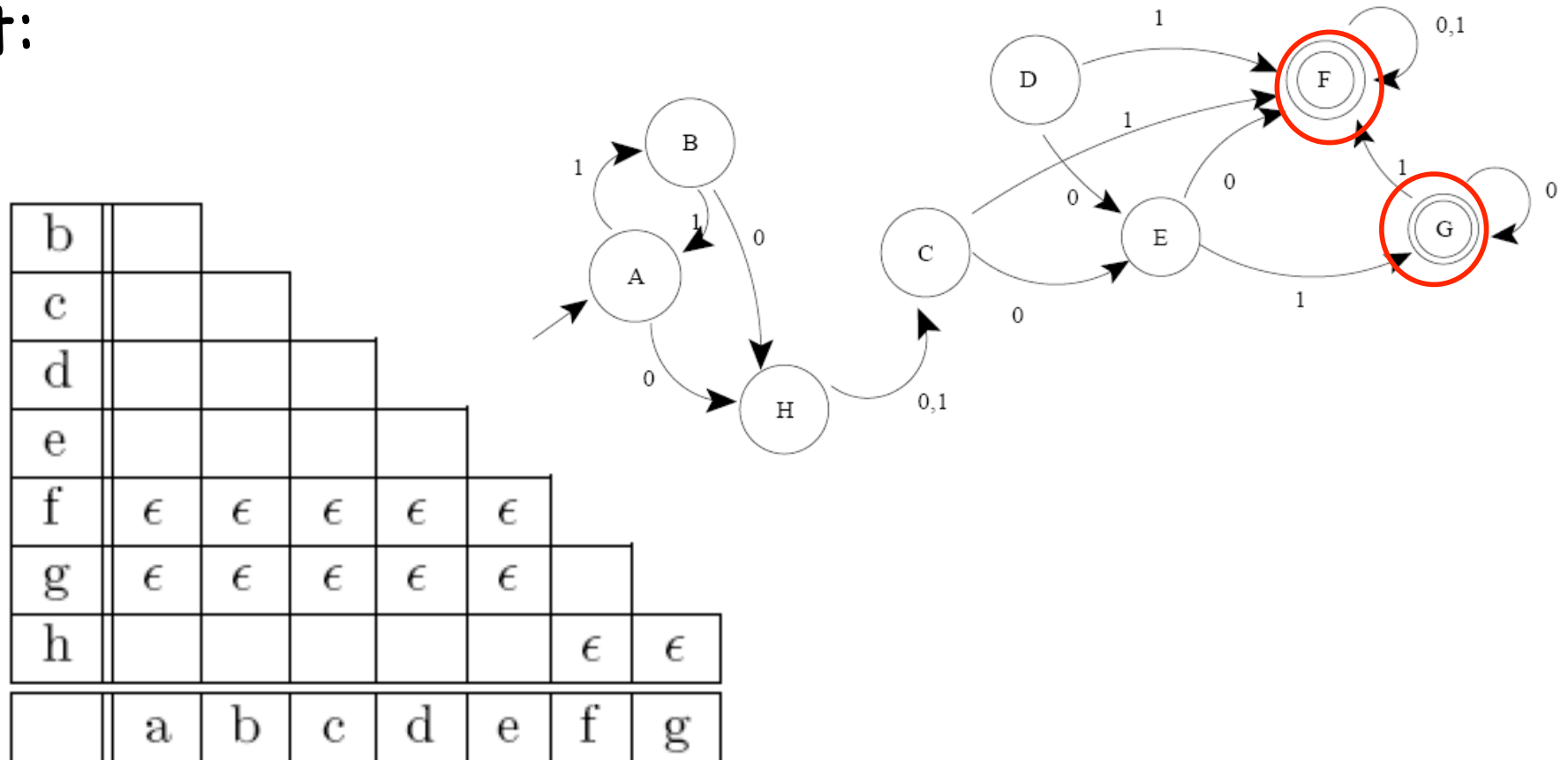


More Complex Example



More Complex Example

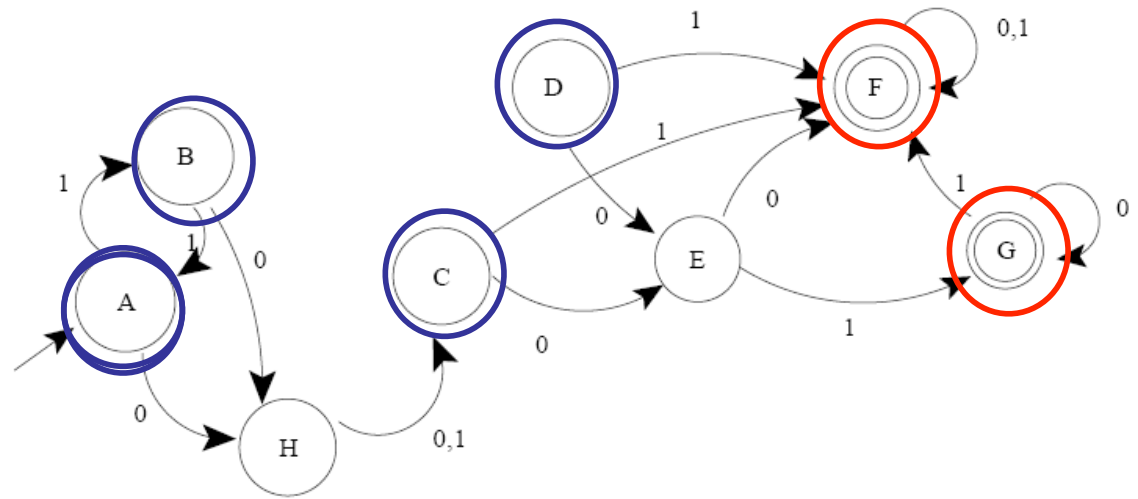
- Check for pairs with one state final and one not:



More Complex Example

- First iteration of main

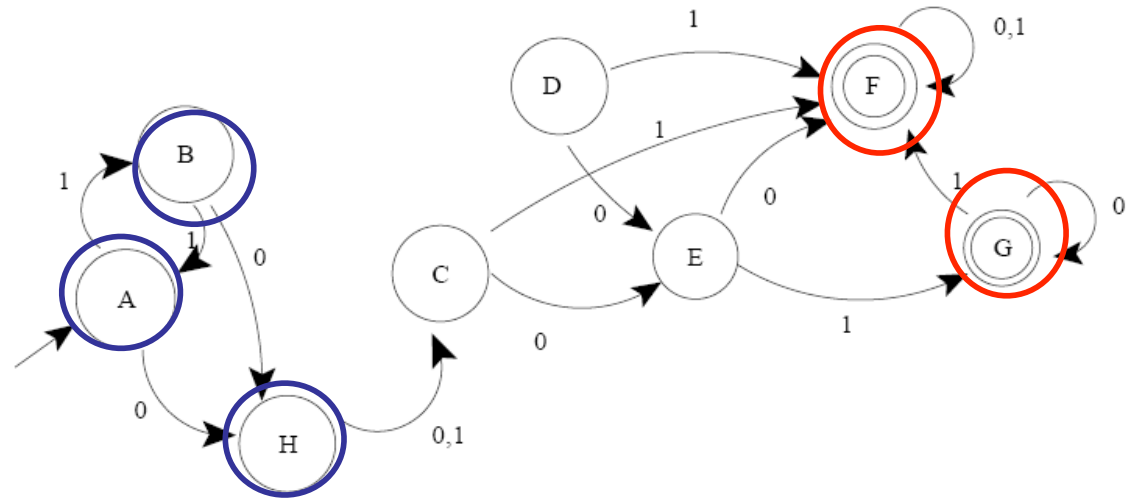
b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h			1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g



More Complex Example

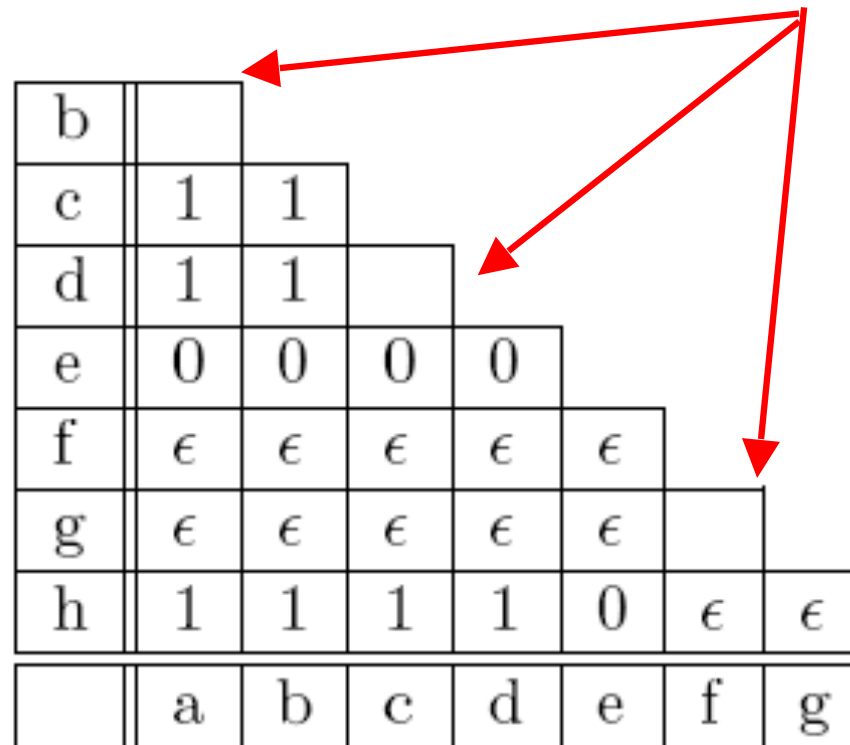
- Second iteration of main loop:

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

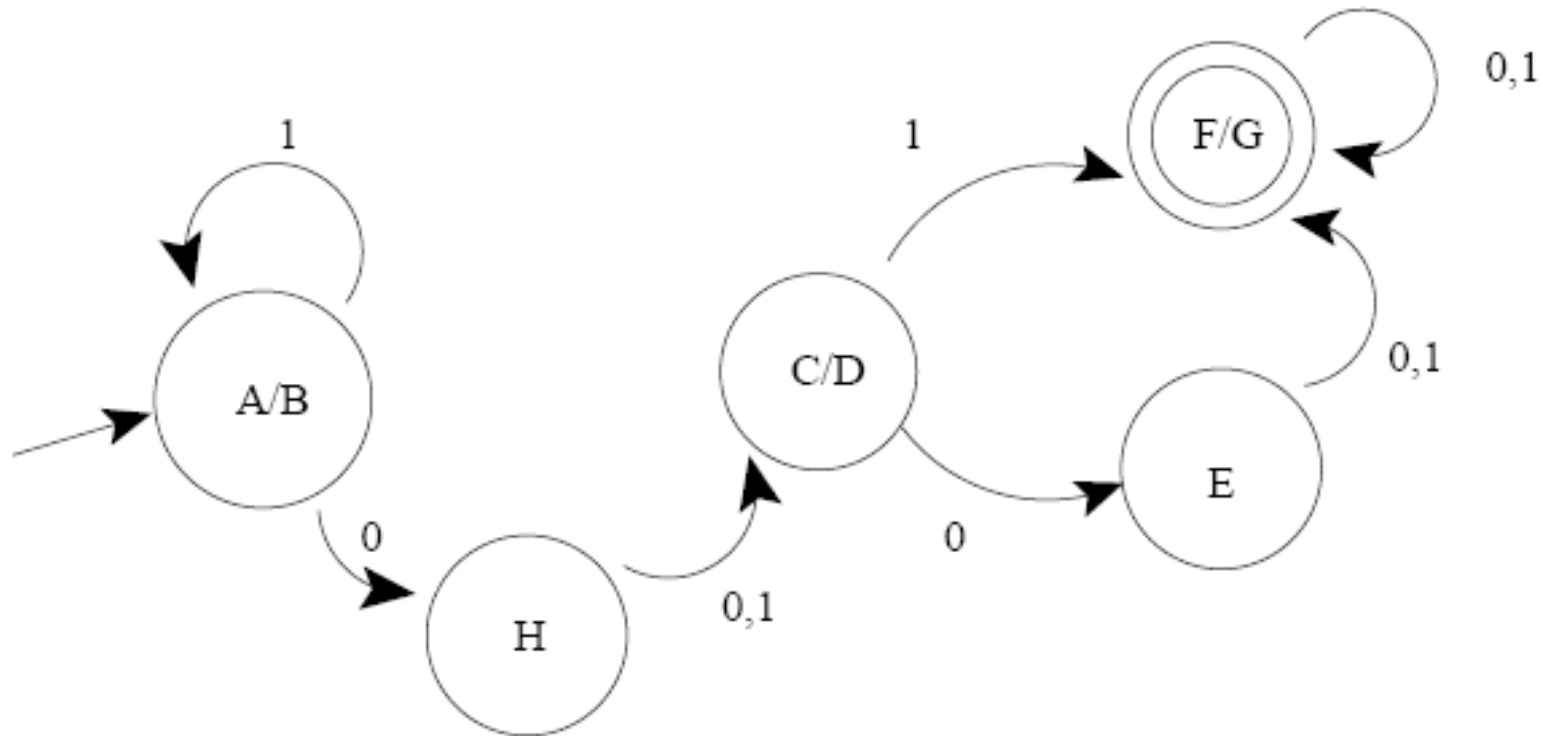
- Third iteration makes no changes
 - Blank cells are equivalent pairs of states



	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

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

$E \mapsto 0$

$E \mapsto 1$

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

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

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

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

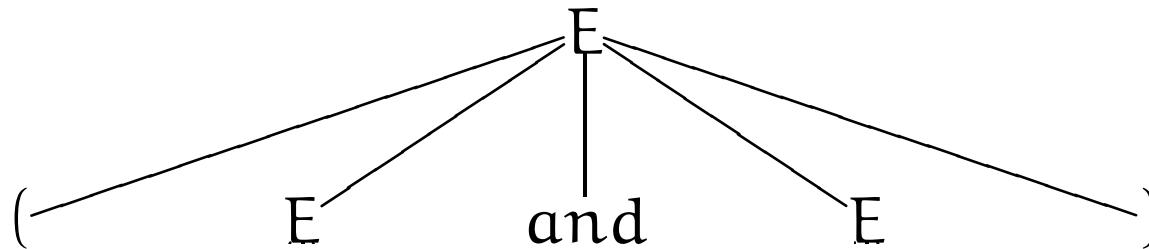
Given a grammar (Σ, N, S, P) .

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 has no children.

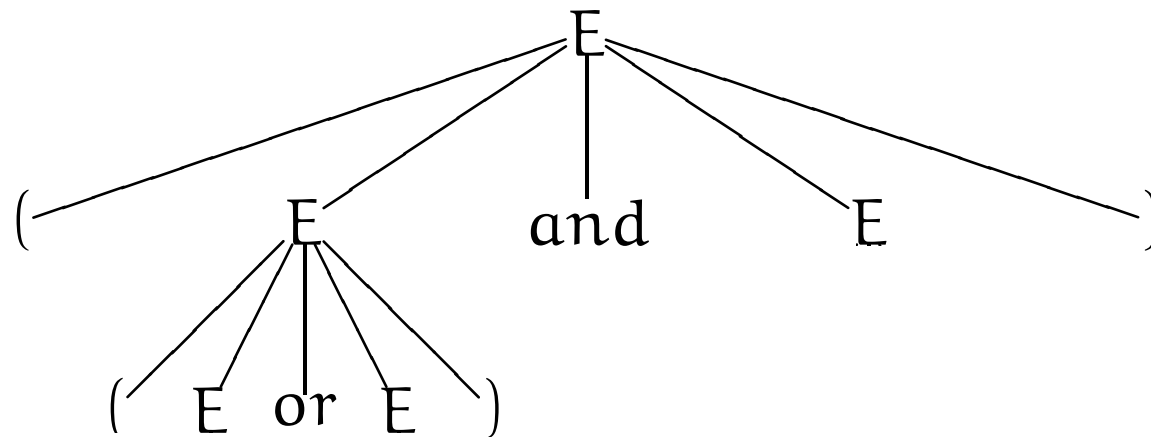
Example

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



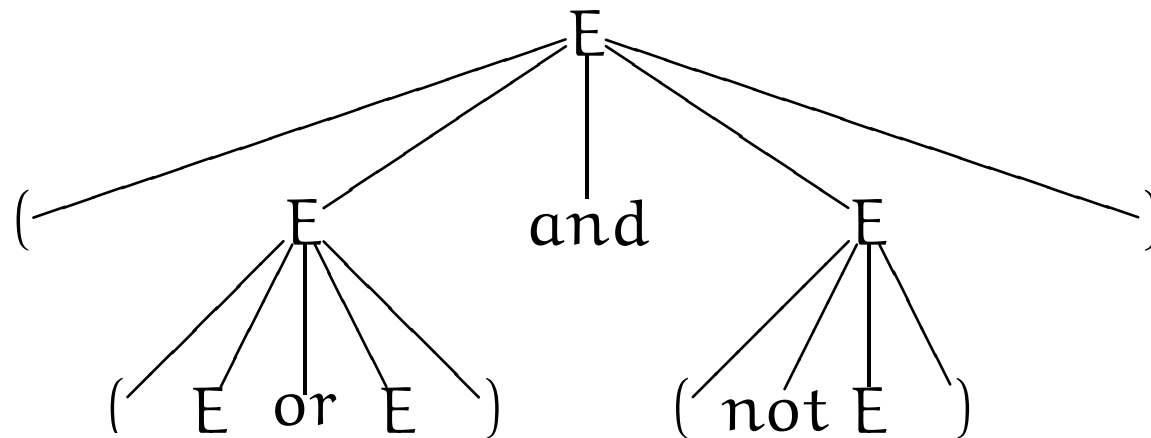
Example

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



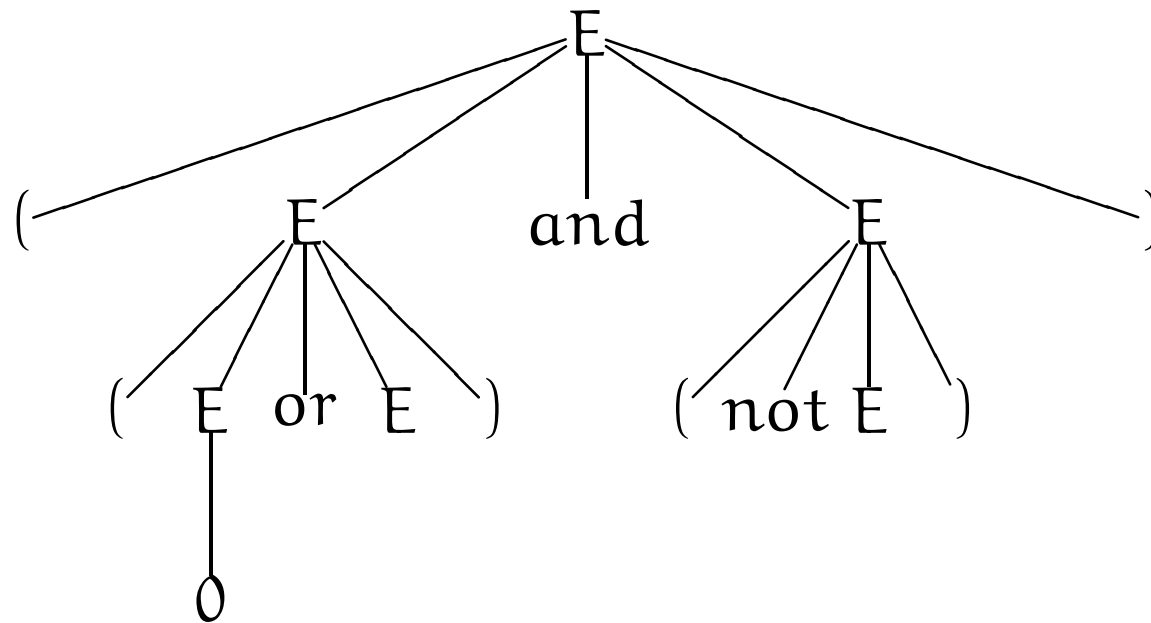
Example

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



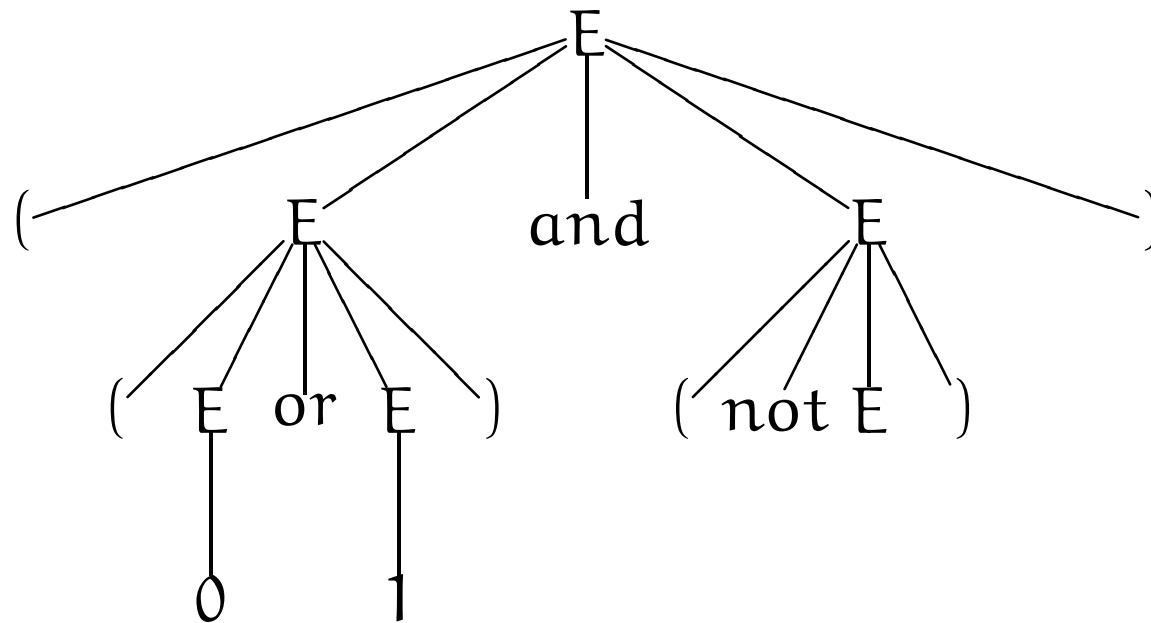
Example

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



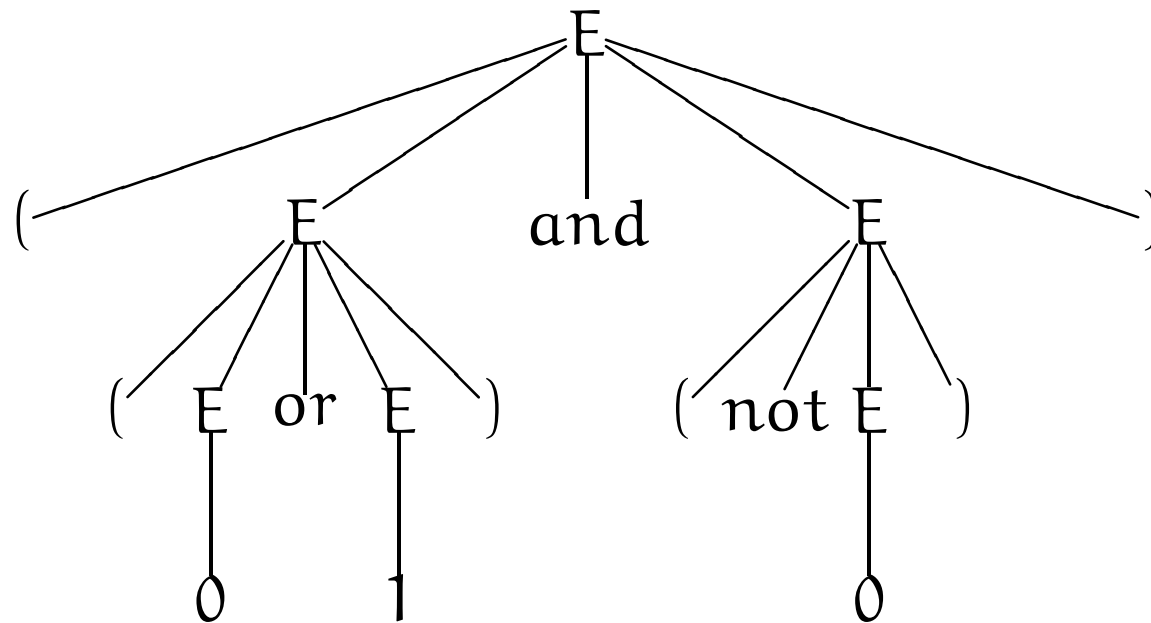
Example

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



Example

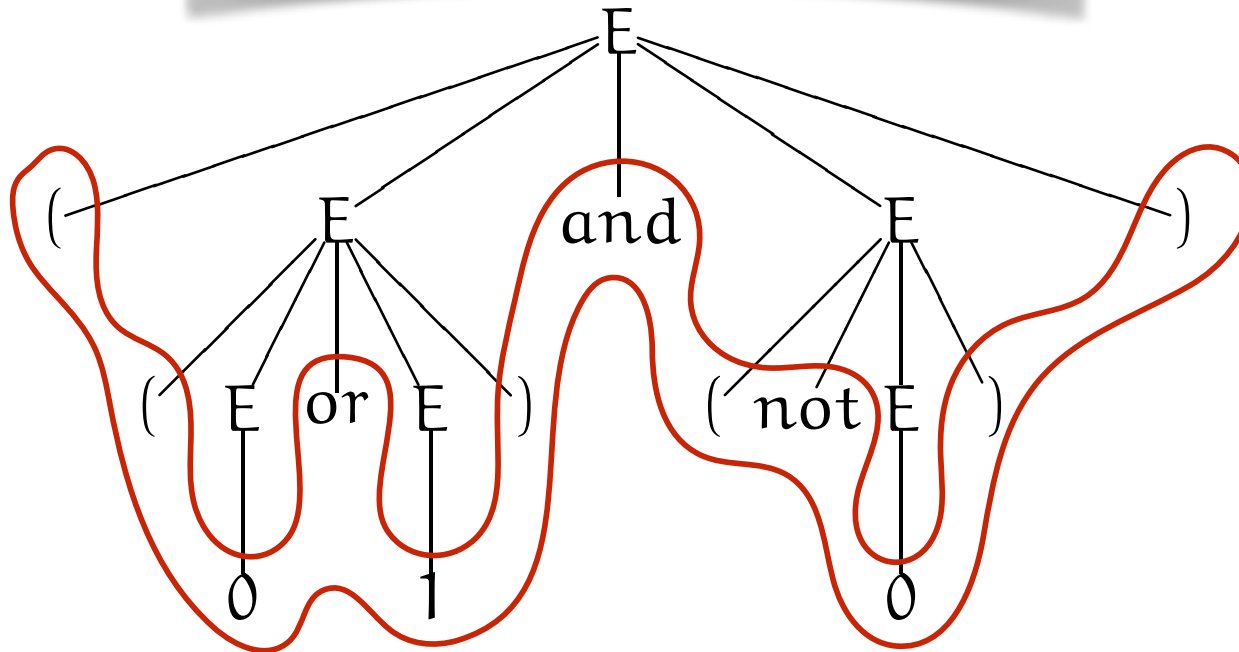
$E \mapsto 0|1|(E \text{ or } E)|(E \text{ and } E)|(\text{not } E).$



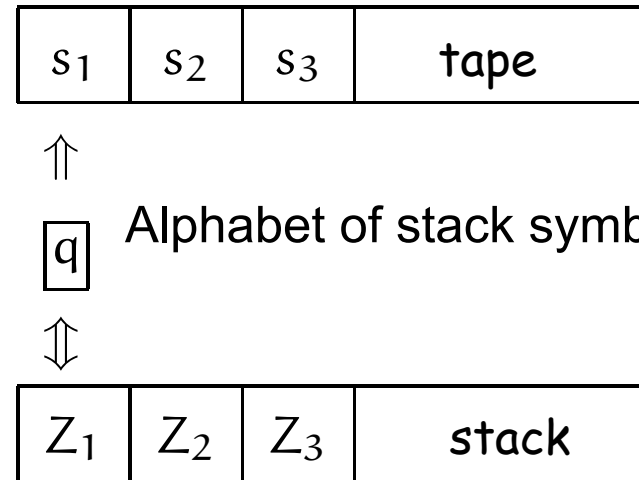
Example

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

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



Pushdown automata



Alphabet of stack symbols: R

The stack head always scans the top symbol
It performs three basic operations:

Push: add a new symbol at the top. of the stack

Pop: read and remove the top symbol

Empty: verify if the stack is empty

Pushdown automata

They can be represented by $M = (Q, \Sigma, R, \delta, q_0, Z_0, F)$ where

- R is the alphabet of stack symbols,
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times R \rightarrow \wp_f(Q \times R^*)$ 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 of the control unit
- w is the unread part of the input string or the remaining input
- γ is the current contents of the PDA 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 a language:

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

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

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

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*

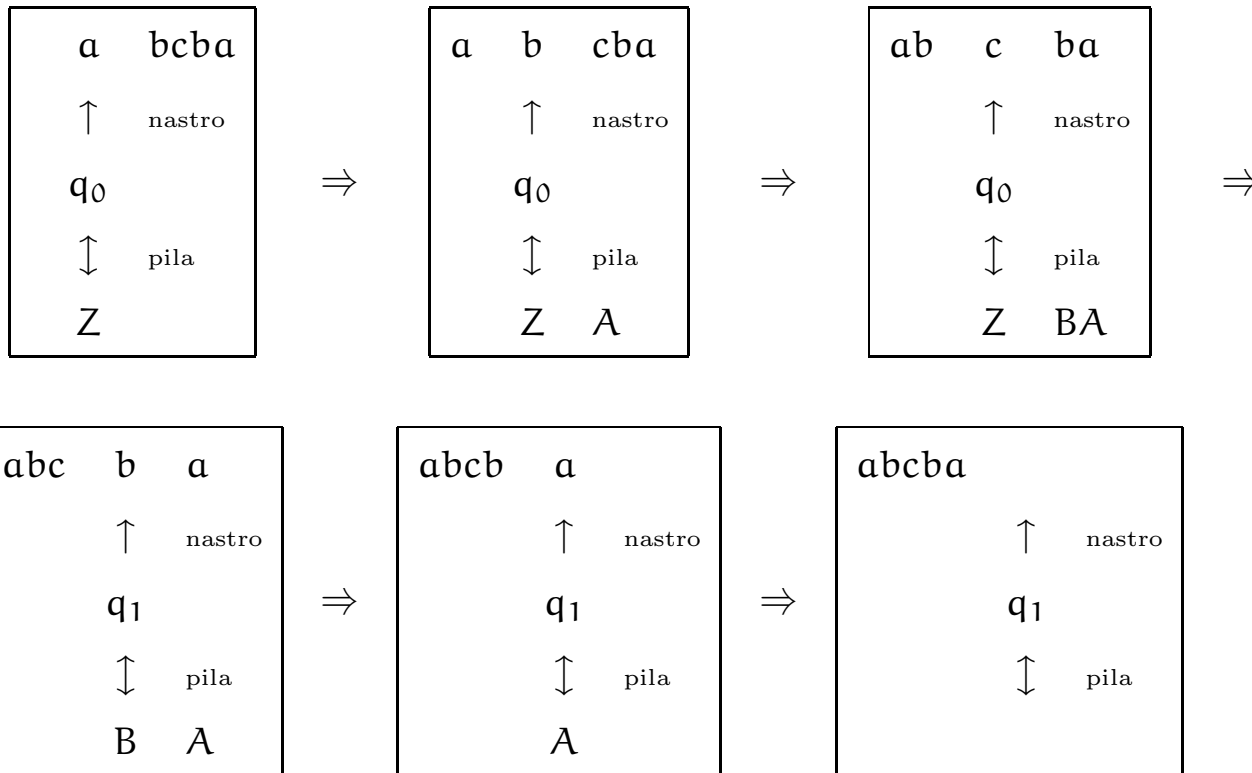
q_0	ε	a	b	c
Z		q_0, ZA	q_0, ZB	q_1, ε
A				
B				

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

Example: abcba

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

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

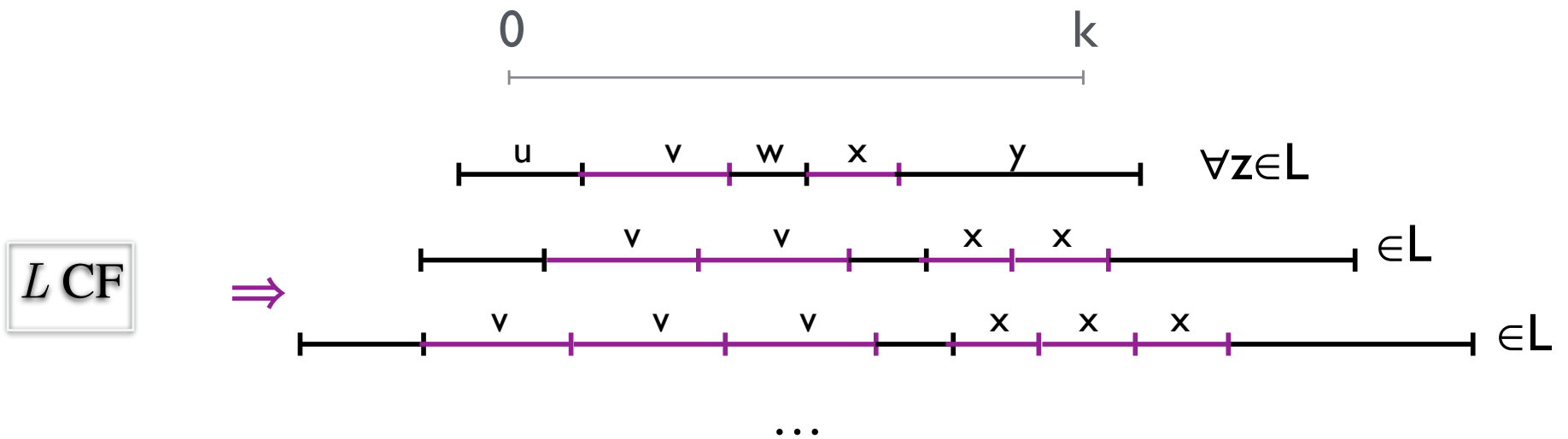
Unfortunately...

not all languages are Context Free !

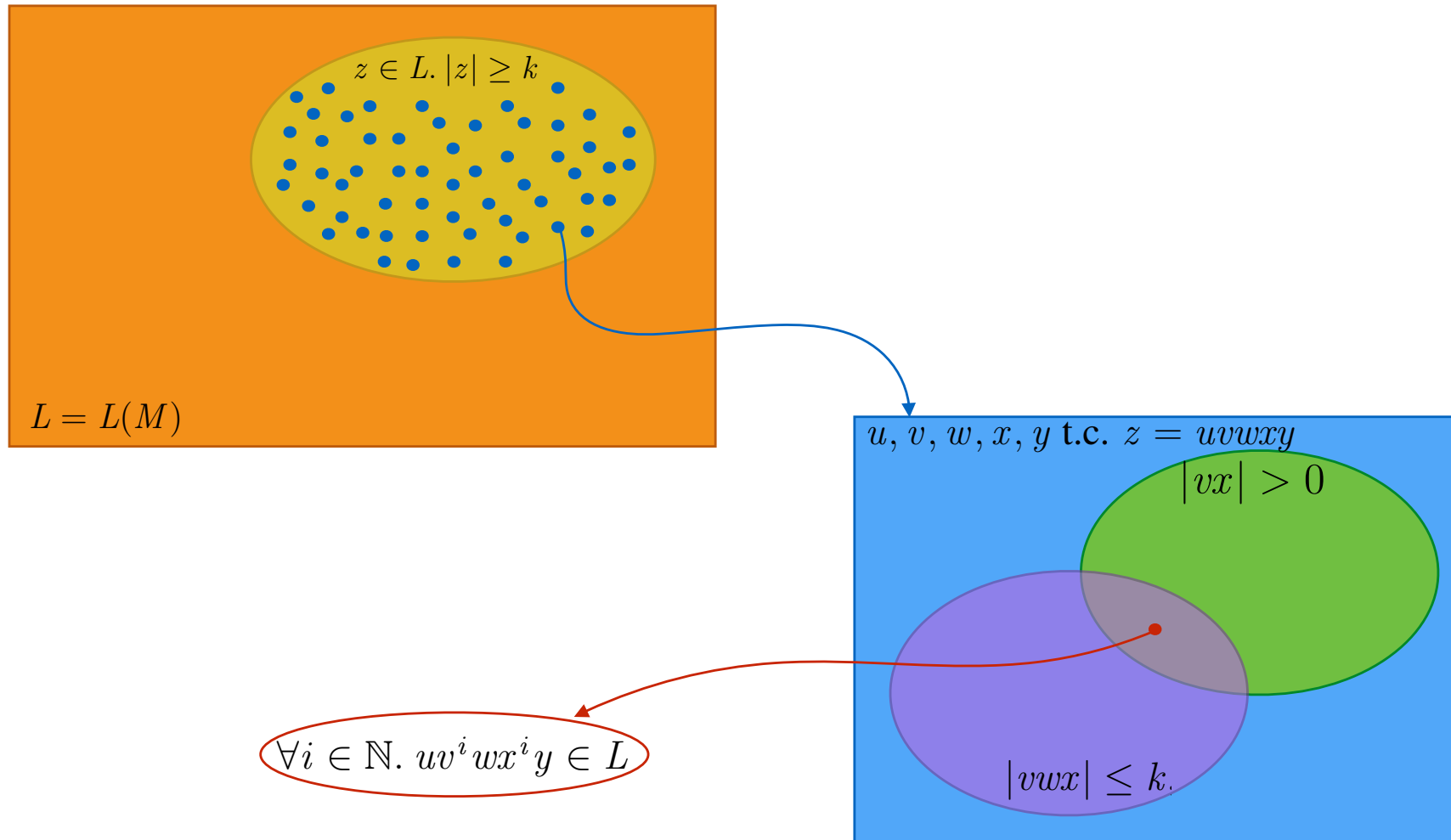
Pumping Lemma for CF

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

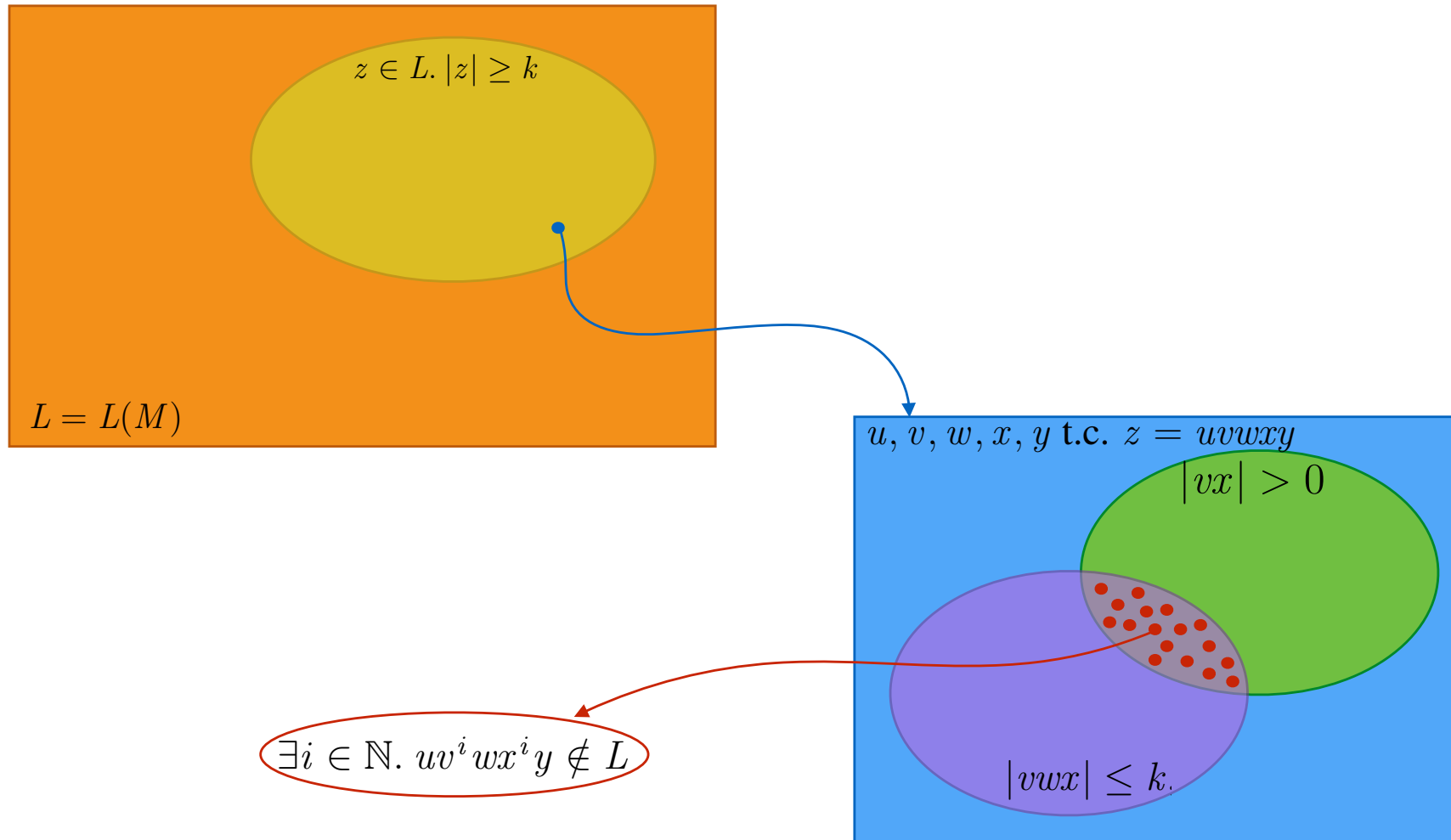
$z = uvwxy$ with $|vwx| \leq k, |vx| > 0$ such that $\forall i \in \mathbf{N}, uv^iwx^iy \in L$



Meaning of PL



Negating the PL



Negating the PL for CF

The PL for CF gives a necessary condition, that can be used to prove that a language **is context free!**

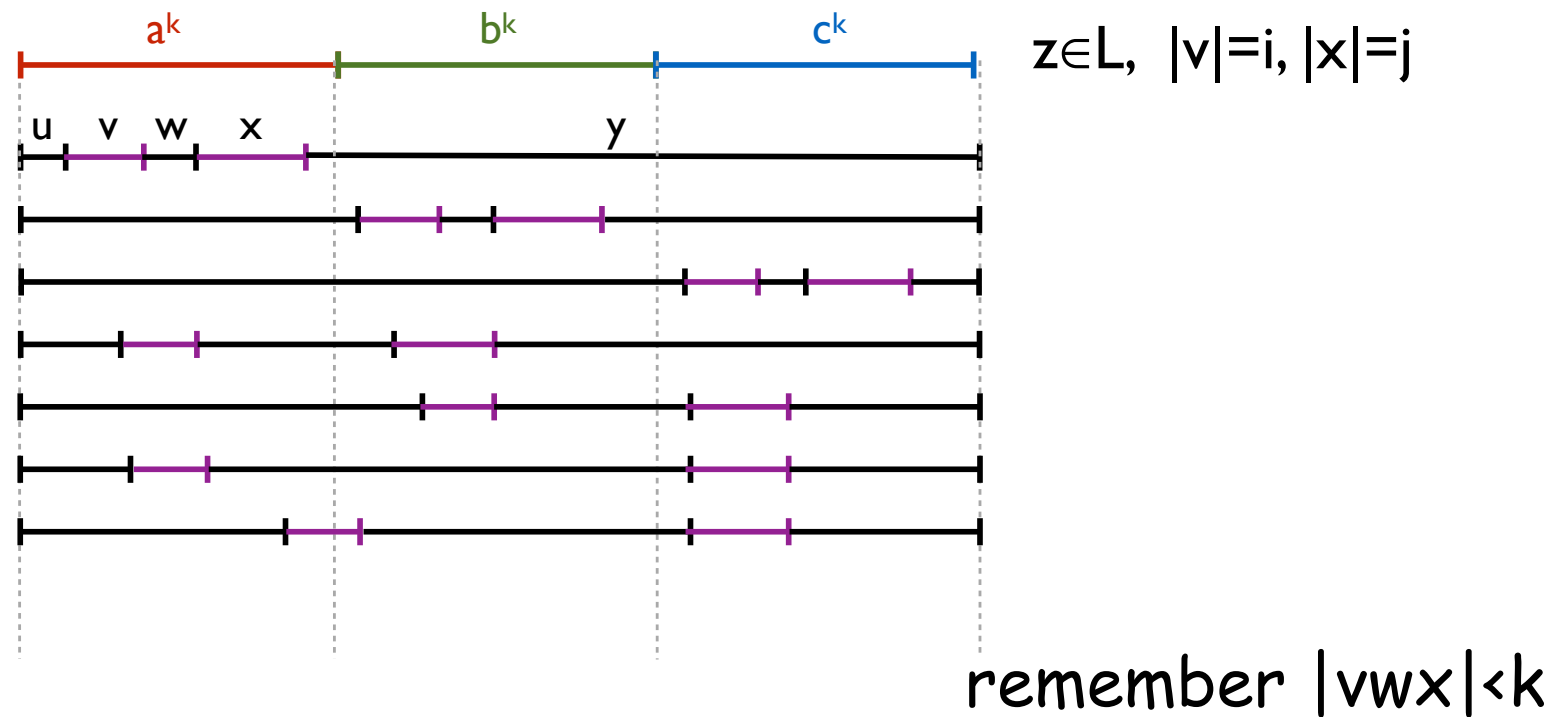
If $\forall k \in \mathbf{N} \exists z \in L. |z| \geq k$ for all possible splitting of the form

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

then **L is not context free!**

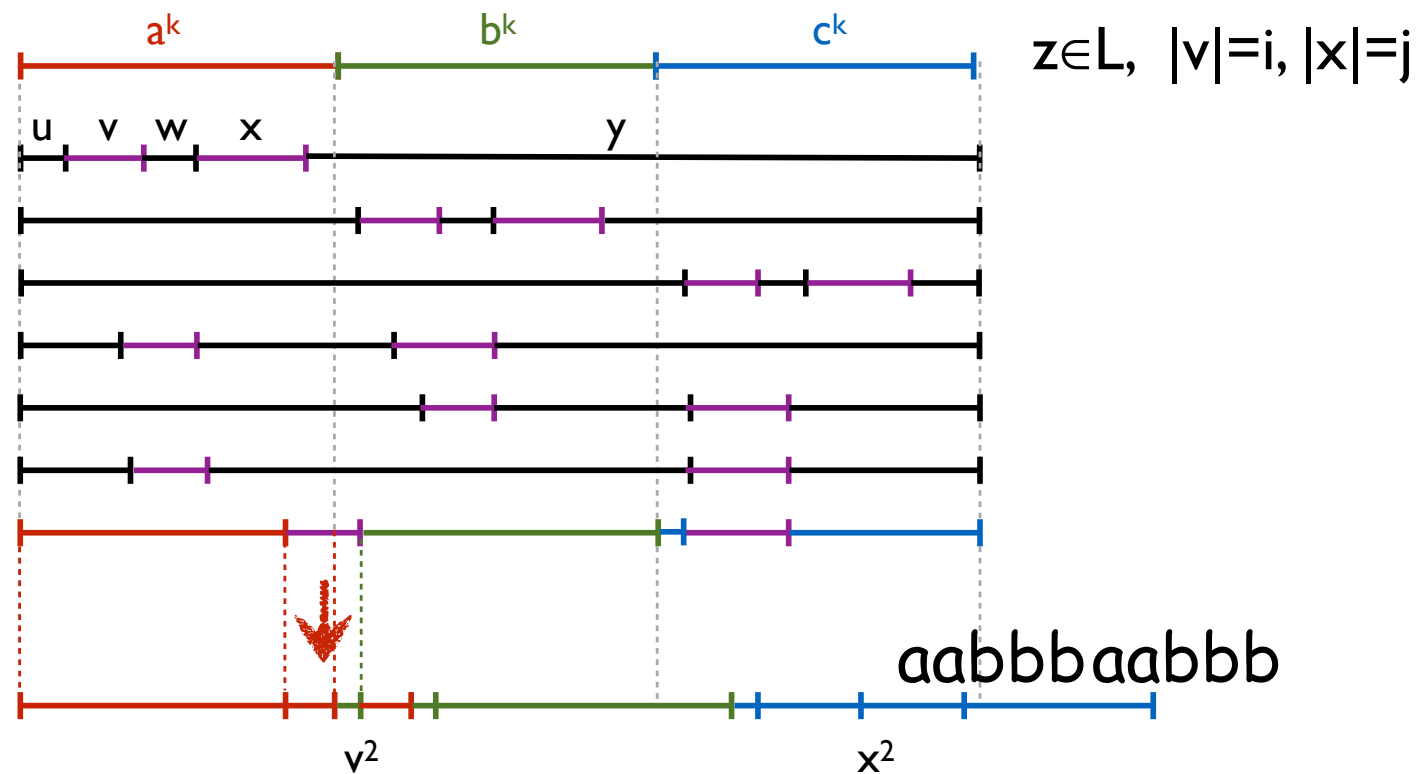
Example

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



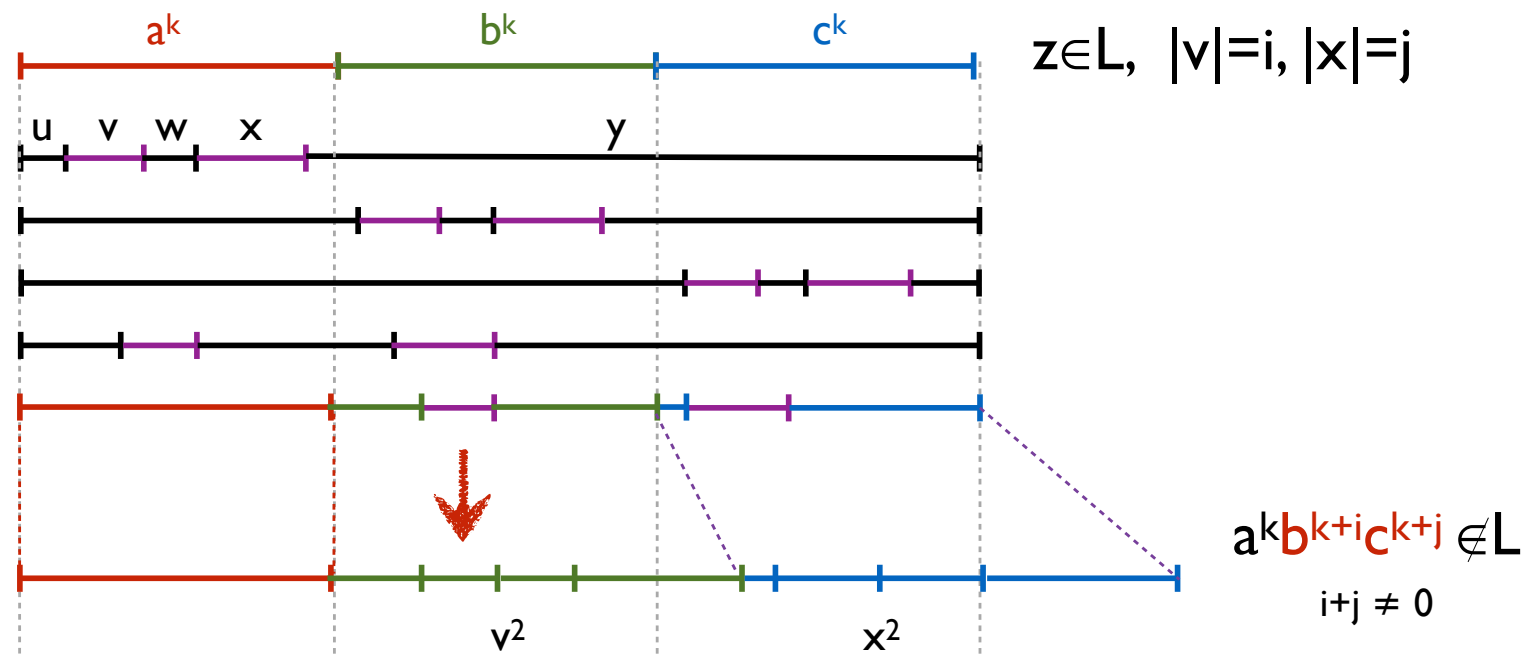
Example

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



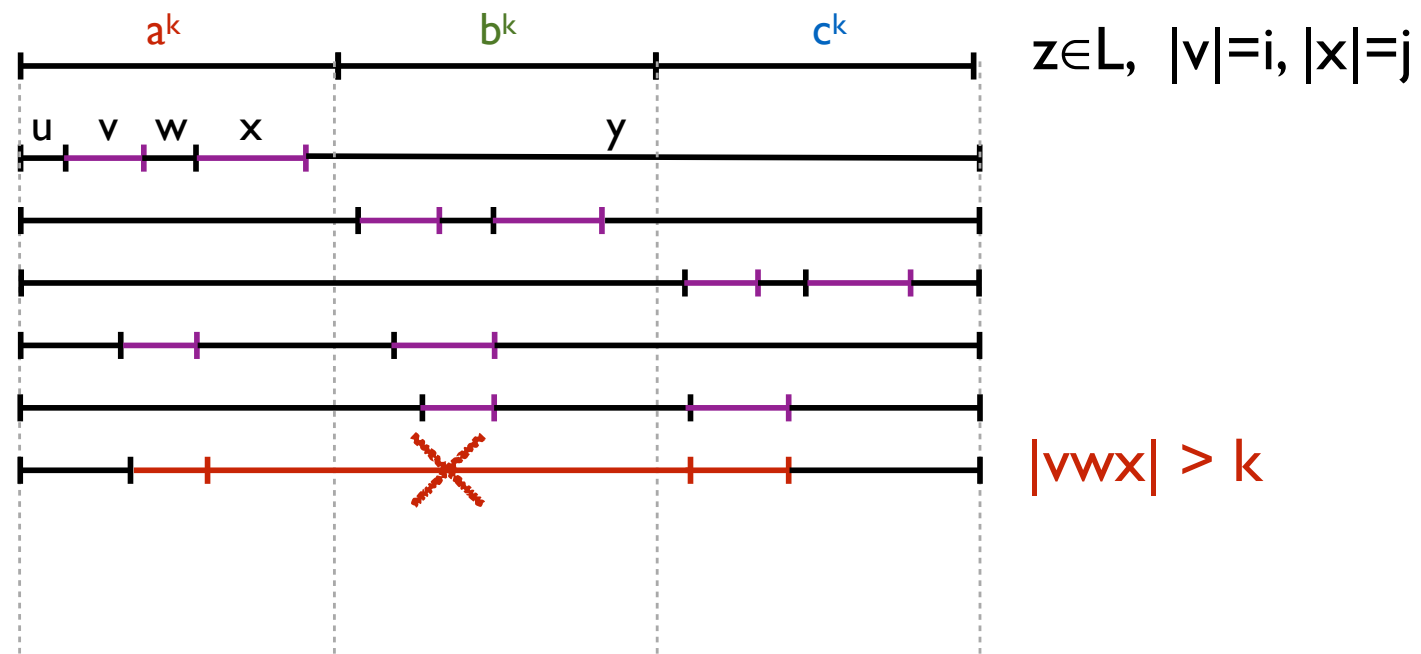
Example

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



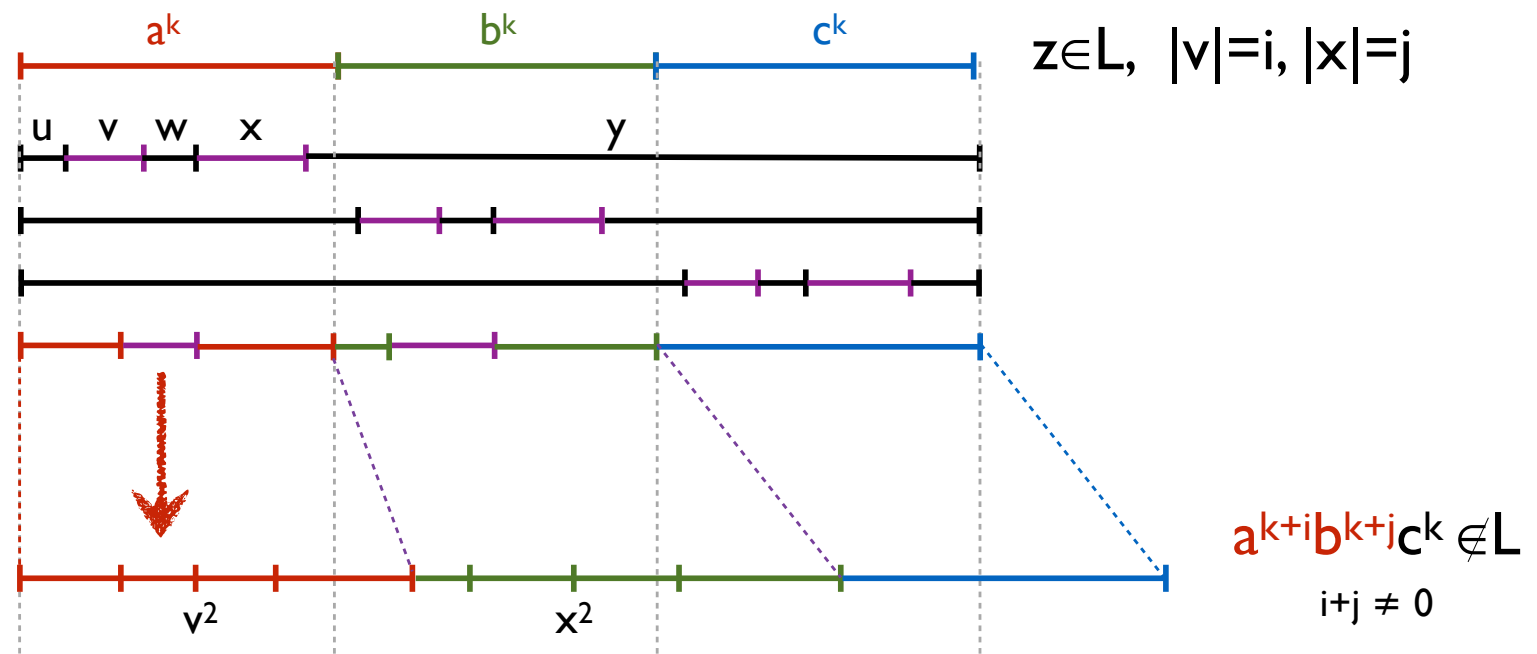
Example

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



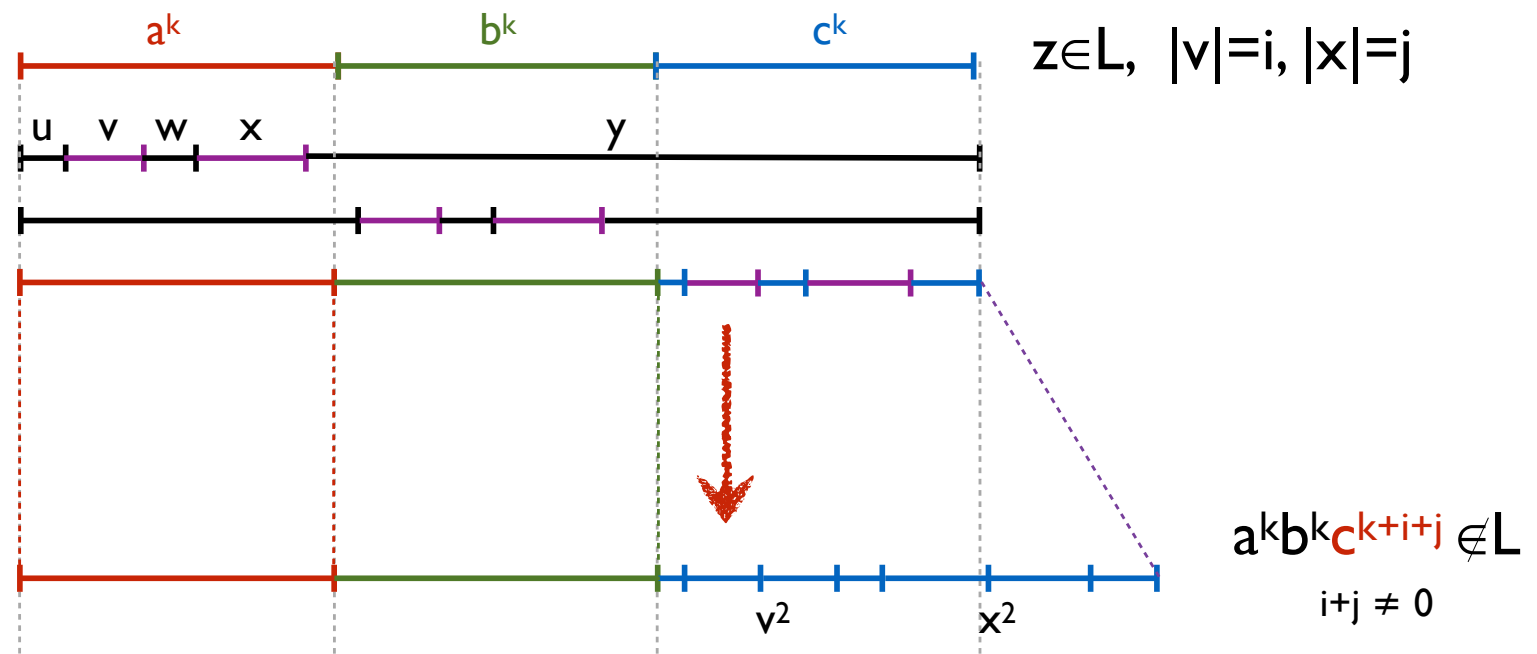
Example

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



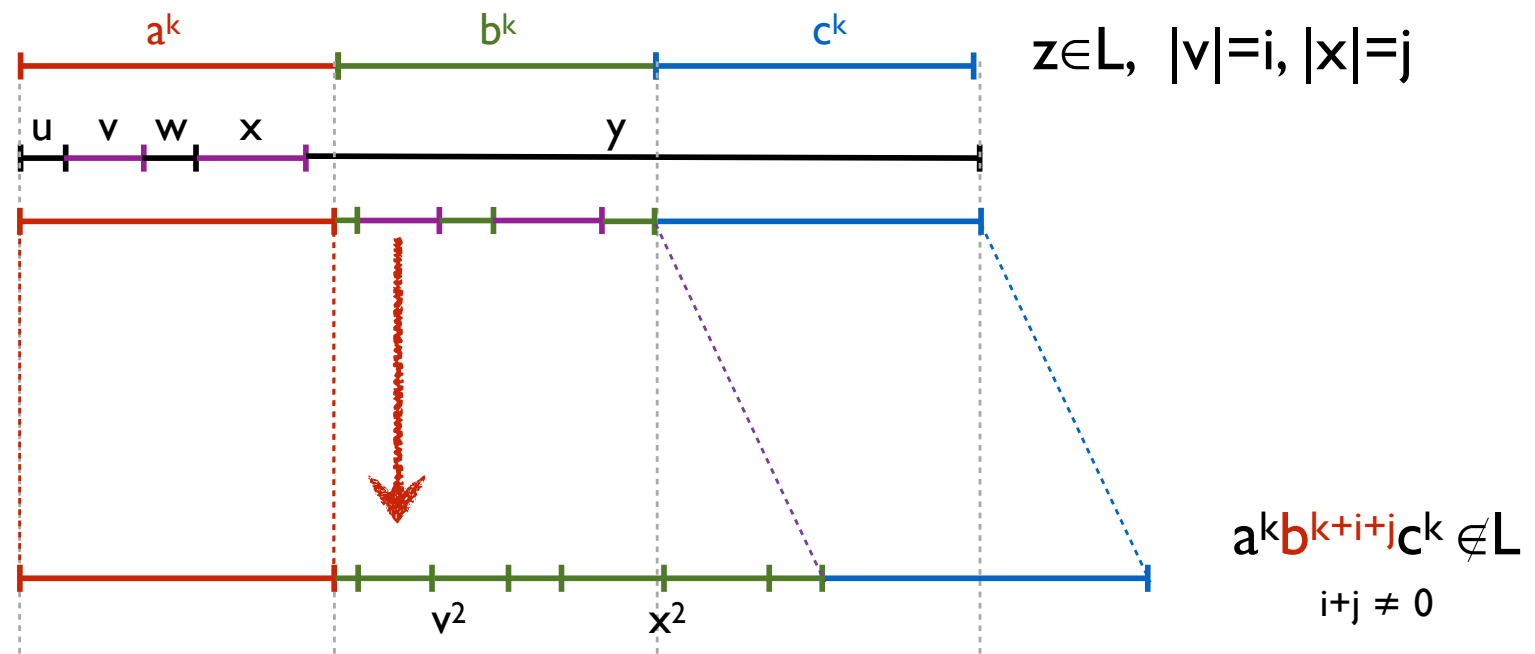
Example

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



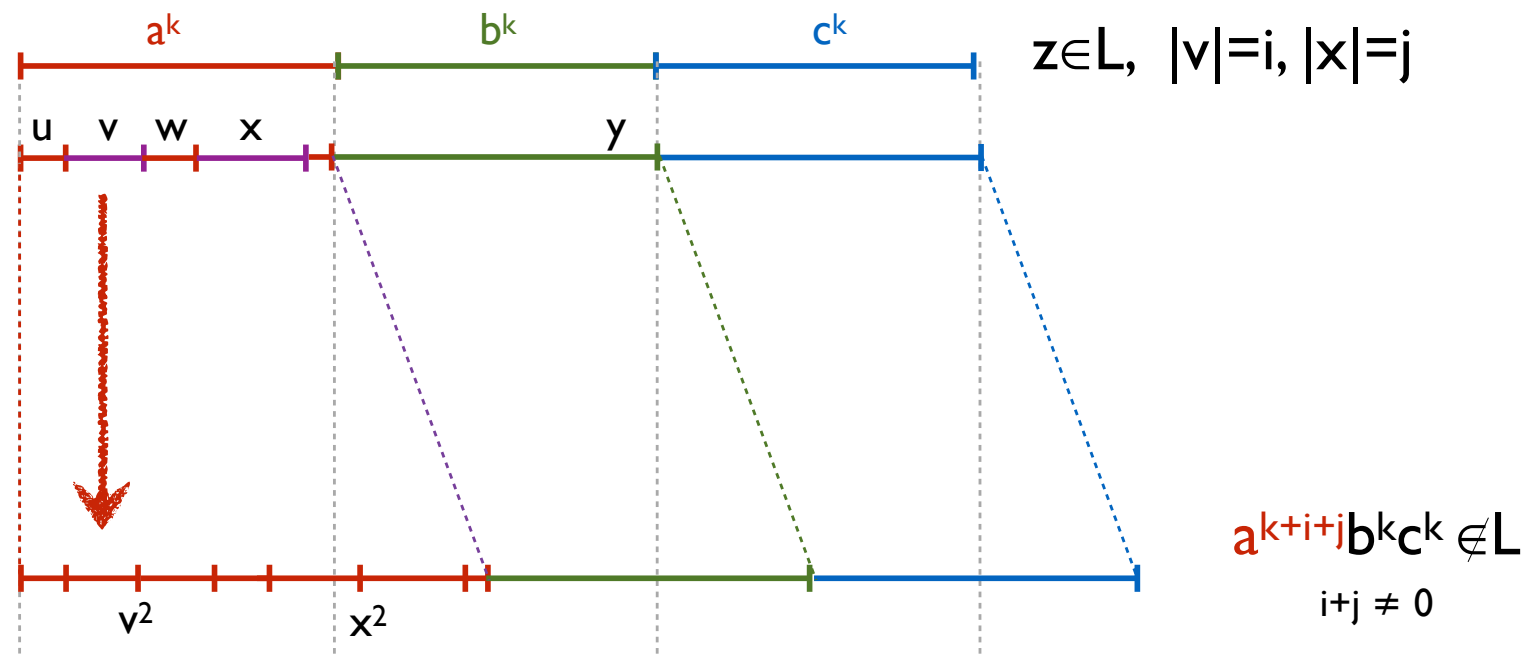
Example

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



Example

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



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

L_1 = strings over $\{0, 1\}$ with an even number of 1's is regular

$L_2 = \{a^n b^n \mid n \geq 0\}$ is context-free, but not regular

$L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is context-sensitive, but not context-free

Relationships between Languages and Automata

A language is :

regular

context-free

context-sensitive

phrase-structure

iff accepted by

finite-state automata

pushdown automata

linear-bounded automata

Turing machine

Chomsky's Hierarchy

