

Non-Deterministic Automata and Language Recognition

Translated and adapted by L. Breveglieri

NON-DETERMINISTIC AUTOMATA (or INDETERMINISTIC)

A right linear grammar may contain alternative rules, i.e., rules from the same state and with the same label, which cause a non-deterministic behaviour

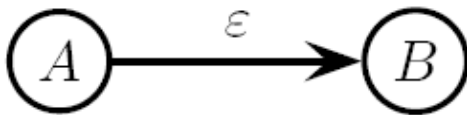


$A \rightarrow a B \mid a C$
where $a \in \Sigma$ and $A, B, C \in V$

$$\delta(A, a) = \{B, C\}$$

the transition function δ of a non-deterministic automaton may have a state set as value (not just a single state)

A non-deterministic behaviour may also originate from a spontaneous move (or ε -move), which is executed without reading any input symbol



$A \rightarrow B$ where $B \in V$

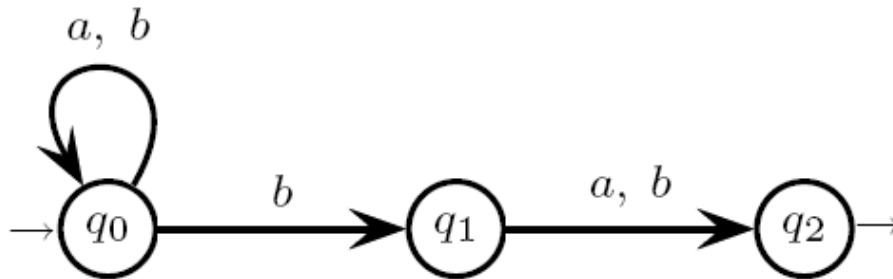


non-deterministic automaton
with a spontaneous move

FOUR MOTIVATIONS FOR INDETERMINISM

- 1) The correspondence between grammars and automata suggests to have
 - a) moves with two or more destination states
 - b) spontaneous moves (or ε -moves)
 - c) two or more initial states, i.e., the grammar has two or more axiomsPoints (a) and (b) are the two main causes of non-determinism in automata
- 2) Concision: defining a language by means of a non-deterministic automaton may be more readable and compact than using a deterministic one

EXAMPLE (all the strings must have a second last character = b)



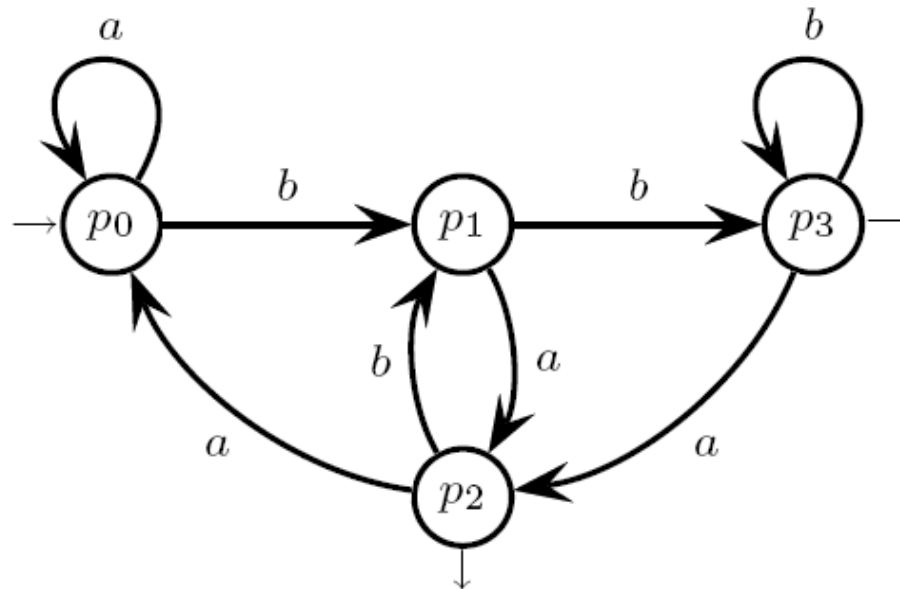
equivalent regexp

$$L = (a \mid b)^* b (a \mid b)$$

Processing string “ $b a b a$ ”. The top computation recognizes the string, the bottom one does not, as it does not go to a final state

$$\begin{array}{ccccccc} & b & & a & & b & & a \\ q_0 & \xrightarrow{\quad} & q_0 & \xrightarrow{\quad} & q_0 & \xrightarrow{\quad} & q_1 & \xrightarrow{\quad} & q_2 \\ & b & & a & & b & & a \\ q_0 & \xrightarrow{\quad} & q_0 & \xrightarrow{\quad} & q_0 & \xrightarrow{\quad} & q_0 & \xrightarrow{\quad} & q_0 \end{array}$$

The same language is accepted by the non-deterministic automaton below, which does not make as evident that the second last character of the strings has to be b



EXERCISE

If we generalize the example and we require that the k -th last element ($k \geq 2$) of the accepted string be b , then the non-deterministic automaton has $k + 1$ states, while it can be proved that the minimum number of states of a deterministic one that recognizes the same language is an exponential function of k

Allowing non-determinism may make some language definitions more coincide

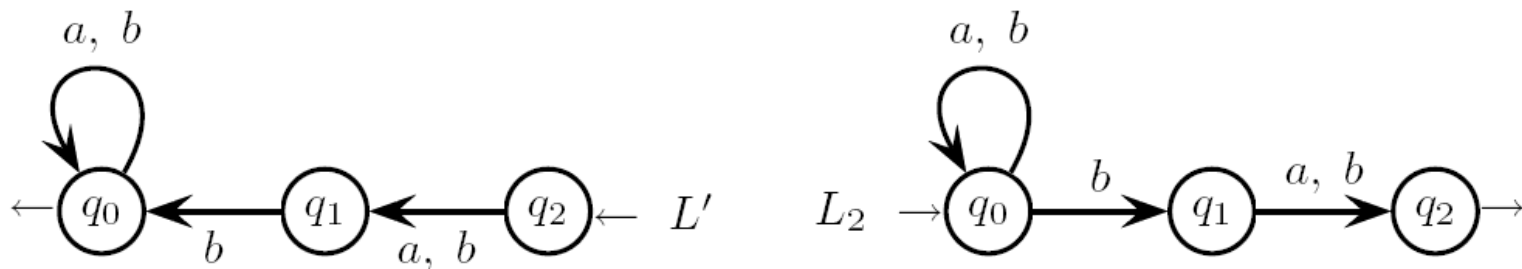
3) Left-to-right duality

Another situation that leads to non-determinism occurs when passing from the deterministic recognizer of a language L to that of the mirror language L^R , where all the strings are reversed. In such a case, indeterminism may originate from the pairs of converging moves with the same label, or by swapping the initial and final states, thus having two or more initial states

EXAMPLE: the language of all the strings with second last character equal to b is the mirror image of that of the strings with second character equal to b

$$L' = \{ x \in \{a, b\}^* \mid b \text{ is the } 2^{nd} \text{ character of } x \}$$

$$L_2 = (L')^R$$



4) Passing through a non-deterministic automaton may be convenient to construct the deterministic automaton equivalent to a regular expression

NON-DETERMINISTIC RECOGNITION

At the moment ignore the existence of spontaneous moves. Here is the formal definition of non-deterministic automaton and computation

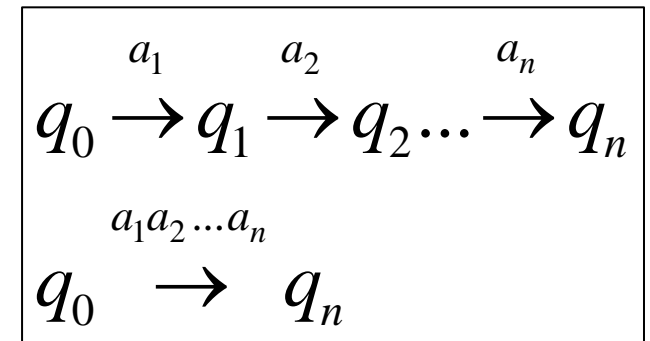
A *non-deterministic finite state automaton* N without spontaneous moves has

1. a finite set of states Q
2. a finite input (or terminal) alphabet Σ
3. two subsets of Q , namely
 - a) the set I of initial states
 - b) the set F of final states
4. a set δ of transitions, which is a subset of the set product $Q \times \Sigma \times Q$

A computation of length n originates at state q_0 and ends at state q_n , and has labeling $a_1 a_2 \dots a_n$

An input string x is accepted (recognized) by the automaton if it is the labeling of a path that starts from an initial state and ends to a final state

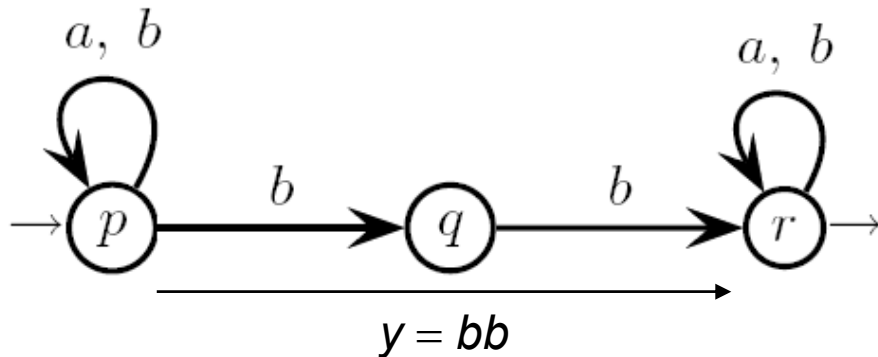
Language recognized by a non-det. automaton N



$$L(N) = \left\{ x \in \Sigma^* \mid q \xrightarrow{x} r \text{ with } q \in I \text{ and } r \in F \right\}$$

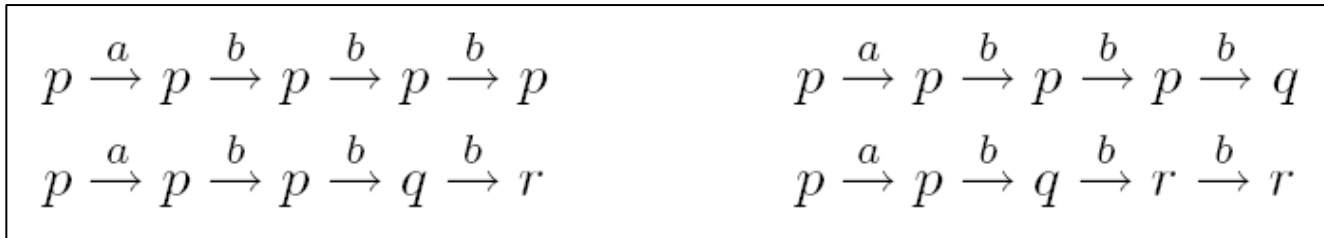
EXAMPLE – searching a word in a text

Given a word y , to recognize if a text contains that word, submit the text to the finite automaton that accepts the (regular) language $(a \mid b)^* y (a \mid b)^*$
 For instance suppose to search the word $y = bb$



the automaton is
non-deterministic
in the state p

The string “ $a \ b \ b \ b$ ” labels several computations that start from the initial state



The two top computations do not find the searched word. The two bottom ones find it in these two positions, respectively

$a \ b \ \underbrace{b \ b}_y$ and $a \ \underbrace{b \ b}_y \ b$

TRANSITION FUNCTION

The moves of the non-deterministic automaton can be defined by means of a many-valued transition function

Given the non-deterministic automaton $N = (Q, \Sigma, \delta, I, F)$ without spontaneous moves, the transition function δ is defined as to have the domain and image

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

For any terminal char. a $\delta(q, a) = \{p_1, p_2, \dots, p_k\}$

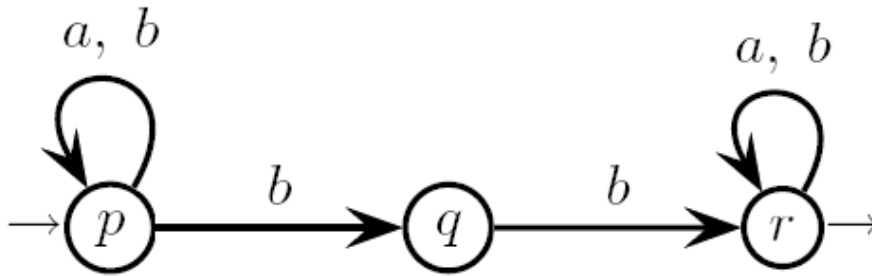
For the empty string ε $\forall q \in Q \quad \delta(q, \varepsilon) = \{q\}$

For any terminal string y $\forall q \in Q \quad \forall y \in \Sigma^* \quad \delta(q, y) = \{p \mid q \xrightarrow{y} p\}$

Here is the language accepted by N : after scanning a string x , if the current state set contains at least one final state, then the string x is accepted

$$L(N) = \{x \in \Sigma^* \mid \exists q \in I \quad \delta(q, x) \cap F \neq \emptyset\}$$

EXAMPLE – searching a word in a text (continued)



$$\begin{aligned}\delta(p, a) &= [p], \\ \delta(p, ab) &= [p, q], \\ \delta(p, abb) &= [p, q, r]\end{aligned}$$

AUTOMATA WITH SPONTANEOUS TRANSITIONS (ε -MOVES)

Spontaneous moves (or ε -moves) are represented by means of arcs labeled with the metasymbol ε (called ε -arcs) in the state-transition graph

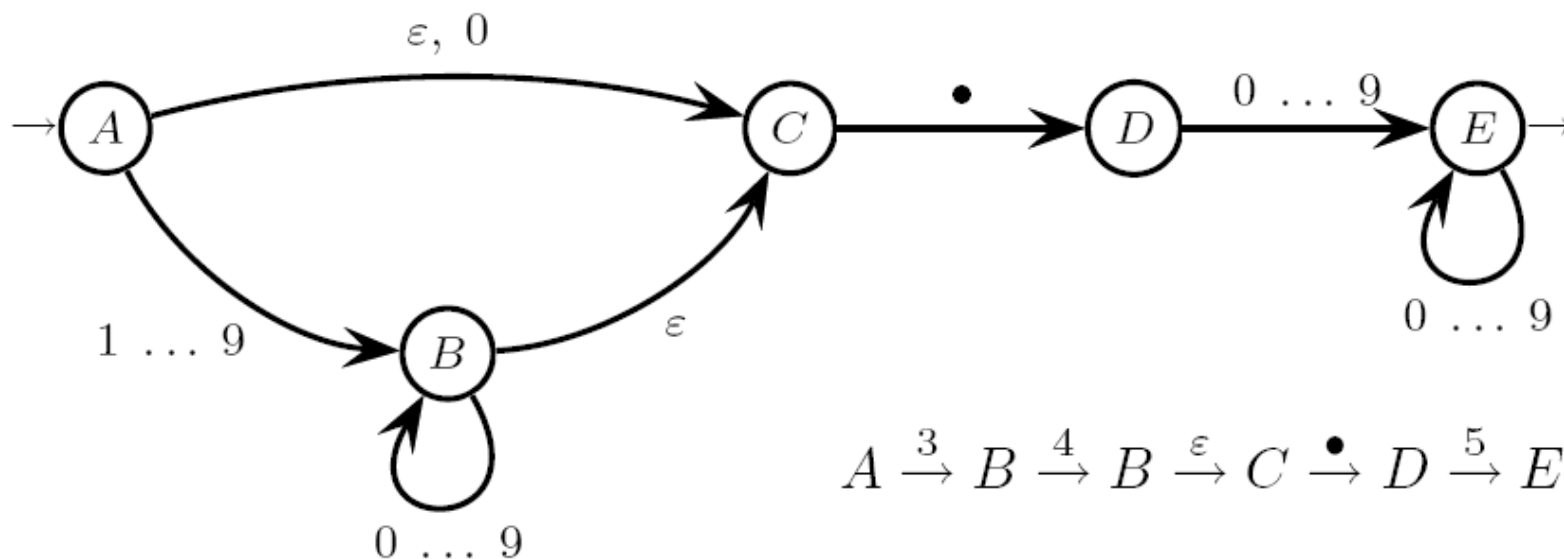
Using ε -arcs helps the modular construction of finite automata and thus allows us to reuse other automata or some parts thereof

EXAMPLE – decimal constant, where union and concatenation are used

$$L = (\varepsilon \mid 0 \mid N) \bullet (0 \dots 9)^+$$

$$\text{where } N = (1 \dots 9) (0 \dots 9)^*$$

non-deterministic
automaton
(with ε -arcs)



The spontaneous moves do not change the definition of string recognition. But the computation may be longer than the string to recognize. For instance the string “3 4 • 5” (of length 4) is accepted by the above computation of 5 steps

UNIQUENESS OF THE INITIAL STATE. A non-deterministic automaton may have two or more initial states. But it is easy to construct an equivalent non-deterministic automaton with only one initial state. Add a new initial state q_0 , connect it to the existing initial states by ε -arcs, make such states non-initial and leave only q_0

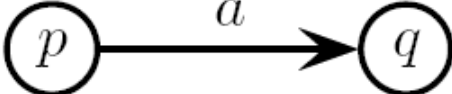
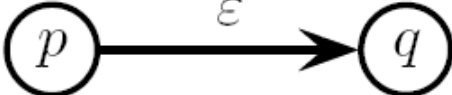
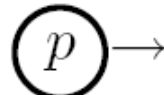
We see soon that a computation of the new automaton accepts a string if, and only if, the old automaton accepts it as well. The additional ε -arcs can be eliminated, as it will be shown next

CORRESPONDENCE BETWEEN AUTOMATON AND GRAMMAR

Suppose that

$G = (V, \Sigma, P, S)$ is a strictly right linear grammar

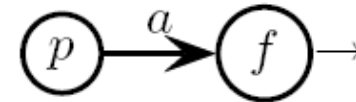
$N = (Q, \Sigma, \delta, q_0, F)$ is the automaton (suppose the initial state is unique)

#	<i>right-linear grammar</i>	<i>finite automaton</i>
1	nonterminal set $V = Q$	set of states $Q = V$
2	axiom $S = q_0$	initial state $q_0 = S$
3	$p \rightarrow a q$ where $a \in \Sigma$ and $p, q \in V$	
4	$p \rightarrow q$ where $p, q \in V$	
5	$p \rightarrow \varepsilon$	final state 

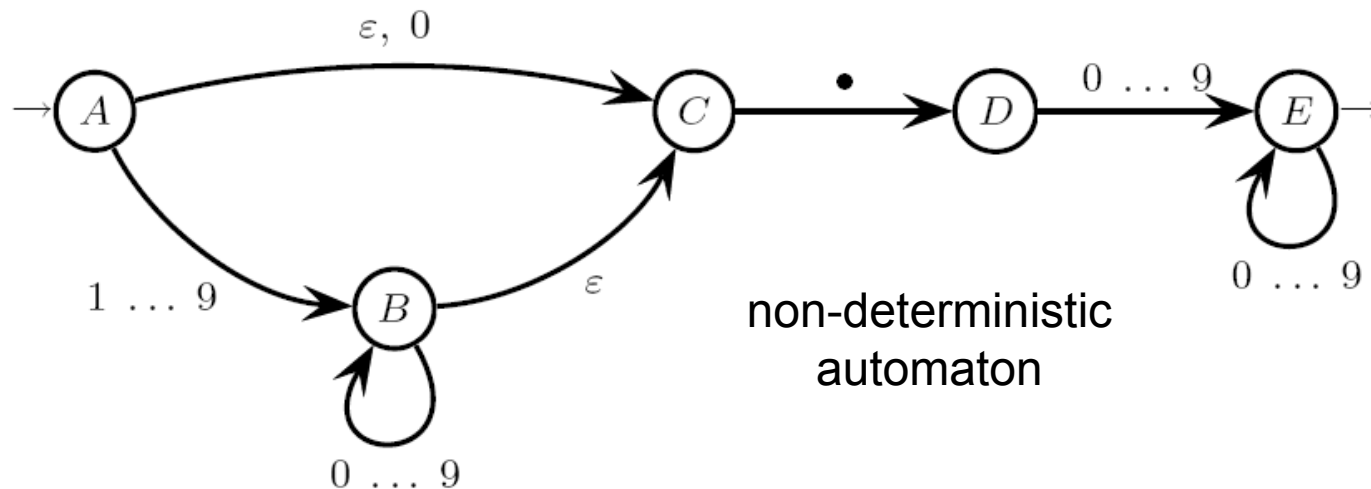
A grammar derivation corresponds to an automaton computation, and viceversa. Consequently the two models define the same language

A LANGUAGE IS GENERATED BY A RIGHT LINEAR GRAMMAR IF AND ONLY IF IT IS RECOGNIZED BY A FINITE AUTOMATON (same for a left linear grammar)

If the (right linear) grammar also has terminal rules of the type $p \rightarrow a$ with $a \in \Sigma$, then the automaton has a final state f in addition to those that correspond to the ε -rules of the grammar, and also has this move



EXAMPLE – equivalence between right linear grammar and automaton

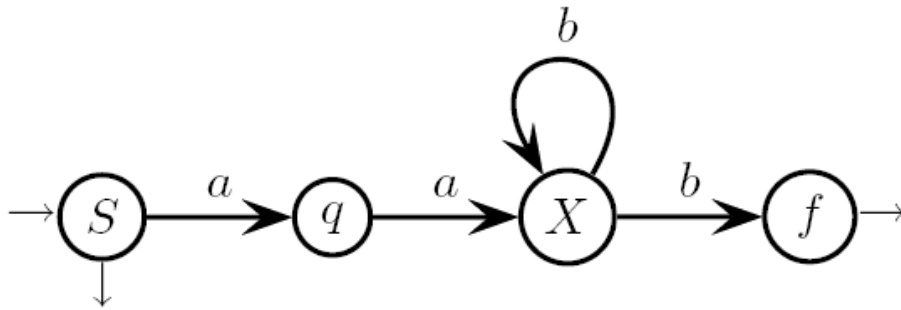


non-deterministic
automaton

$$\left\{ \begin{array}{l} A \rightarrow 0 C \mid C \mid 1 B \mid \dots \mid 9 B \\ C \rightarrow \bullet D \\ E \rightarrow 0 E \mid \dots \mid 9 E \mid \varepsilon \end{array} \right. \quad \begin{array}{l} B \rightarrow 0 B \mid \dots \mid 9 B \mid C \\ D \rightarrow 0 E \mid \dots \mid 9 E \end{array}$$

right linear grammar (axiom A)

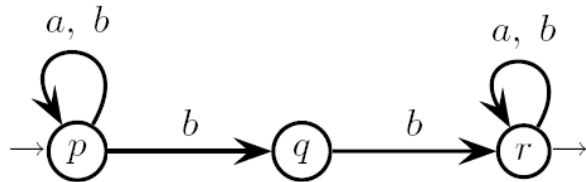
EXAMPLE – right linear grammar, but not strictly linear



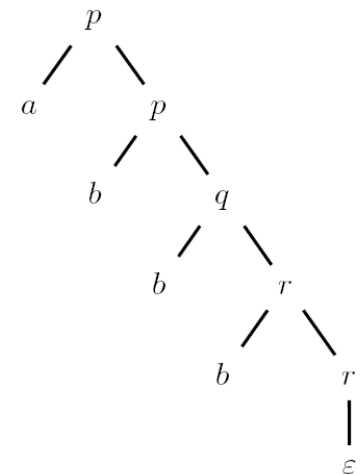
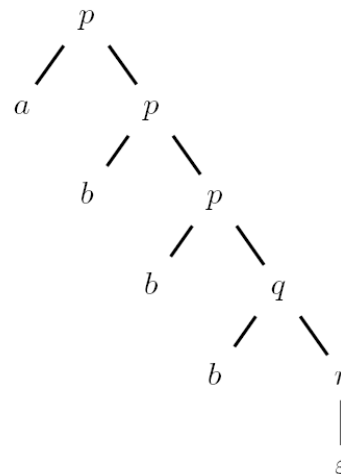
$$\begin{cases} S \rightarrow a a X \mid \varepsilon \\ X \rightarrow b X \mid b \end{cases}$$

AUTOMATON AMBIGUITY – As grammar derivations are in one-to-one correspondence with automaton computations, ambiguity is extensible to automata: an automaton is ambiguous if, and only if, the corresponding grammar is so, i.e., if a string x labels two (or more) accepting paths

EXAMPLE – searching a word in a text – recognize string “ $a b b b$ ”



$$\begin{cases} p \rightarrow a p \mid b p \mid b q \\ r \rightarrow a r \mid b r \mid \varepsilon \\ q \rightarrow b r \end{cases}$$



LEFT LINEAR GRAMMAR AND AUTOMATON

$$A \rightarrow B a$$

$$A \rightarrow B$$

$$A \rightarrow \varepsilon$$

$$L^R = (L(G))^R \text{ is generated by grammar } G_R$$

EXAMPLE – the language of the strings where the second last character is b

$$G: S \rightarrow A a \mid A b$$

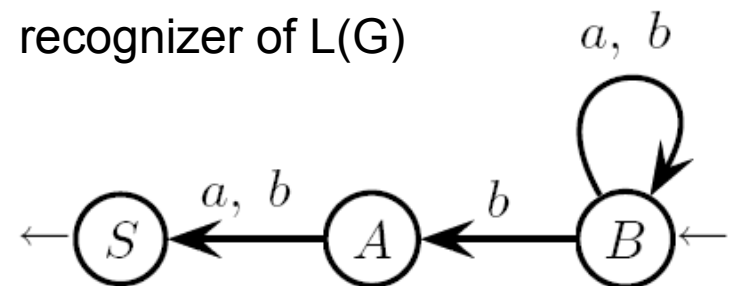
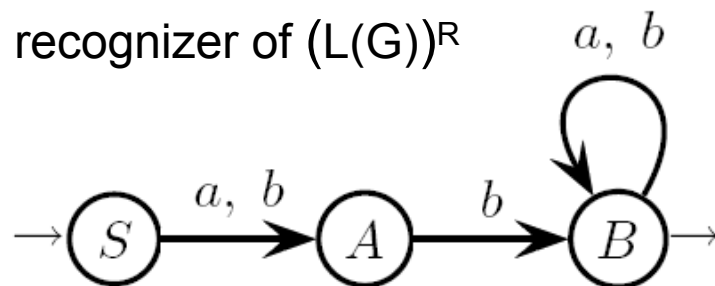
$$A \rightarrow B b$$

$$B \rightarrow B a \mid B b \mid \varepsilon$$

$$G_R: S \rightarrow a A \mid b A$$

$$A \rightarrow b B$$

$$B \rightarrow a B \mid b B \mid \varepsilon$$



FROM THE AUTOMATON TO THE REGEXP DIRECTLY
BMC METHOD (Brzozowski and McCluskey)
ALSO KNOWN AS “NODE ELIMINATION” METHOD

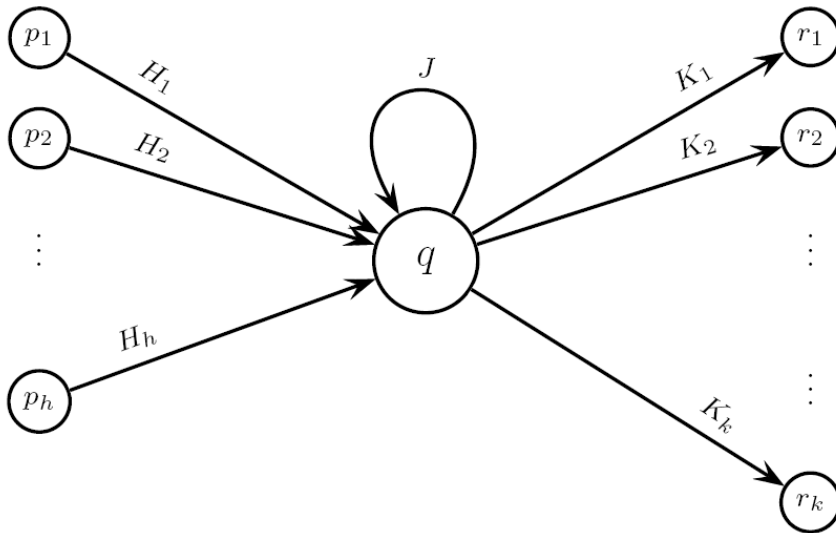
Assume the initial and final states i and t are unique and do not have incoming or outgoing arcs, respectively, i.e., they are not recirculated. If it is not so, then modify the automaton (see the next example)

INTERNAL STATES: all the other states different from i and t

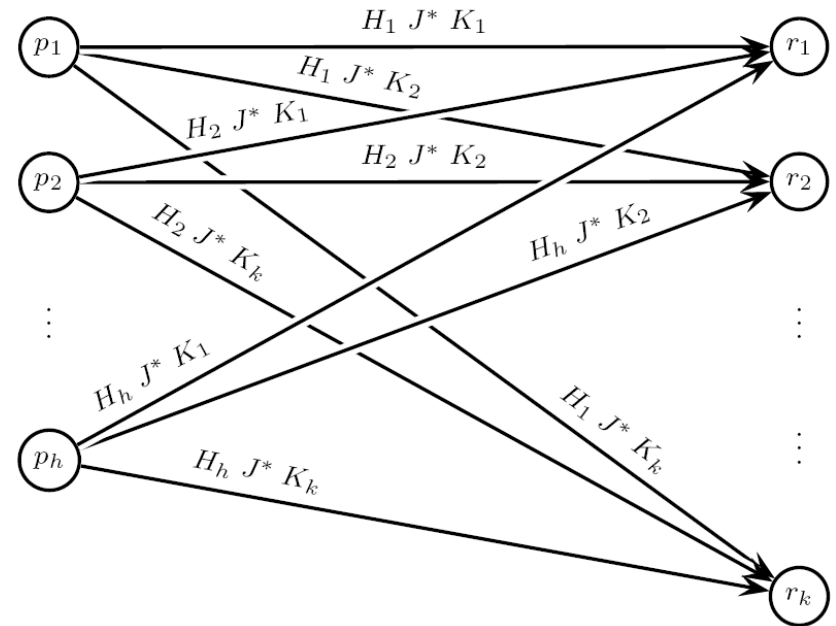
Construct the so-called GENERALIZED FINITE AUTOMATON: its arcs may be labeled with regular expressions, not only with individual terminal characters

Eliminate the internal nodes one by one, and after each elimination add one or more compensation arcs to preserve the equivalence of the automaton. Such new arcs are labeled by regexps. At the end only the nodes i and t are left, with only one arc from i to t . The regexp that labels such an arc generates the complete language recognized by the original finite automaton

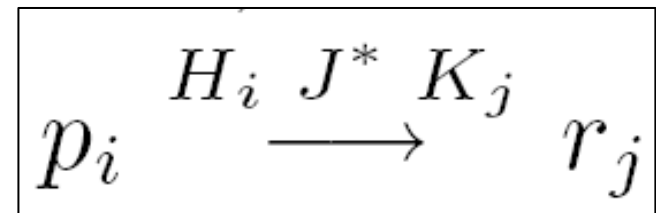
before eliminating node q



after eliminating node q

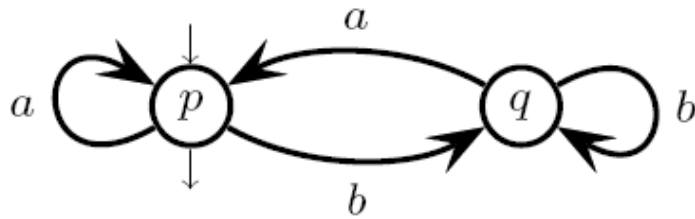


For every state pair p_i, r_j there is the arc;
sometimes p_i and r_j may coincide (self-loop)

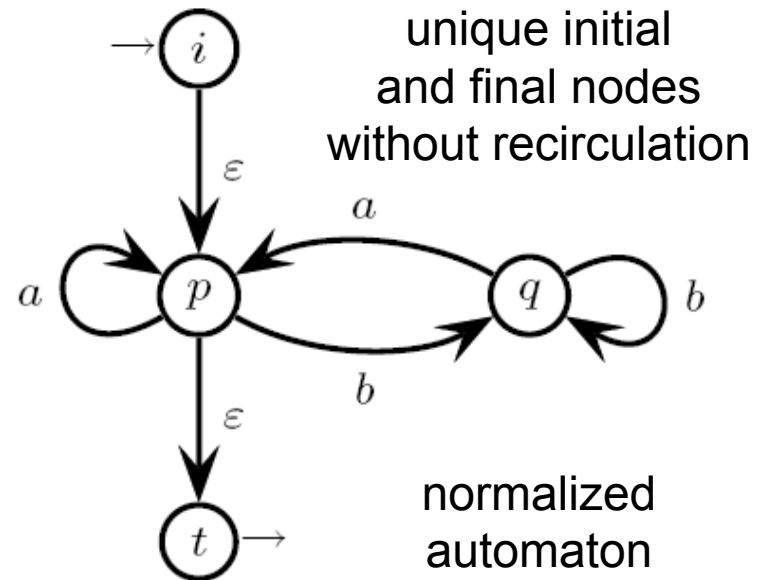


The elimination order is not relevant. However different orders may generate different regexps, all equivalent to one another but of different complexity

EXMPLE – automaton normalization and then node elimination in the order q, p

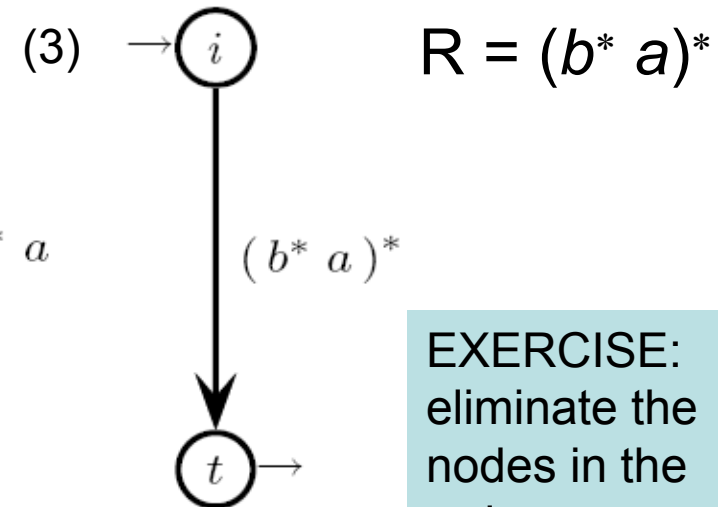
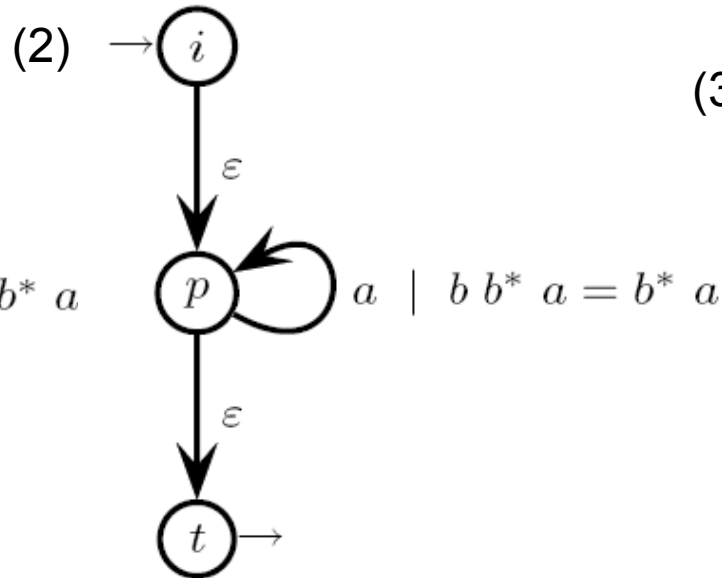
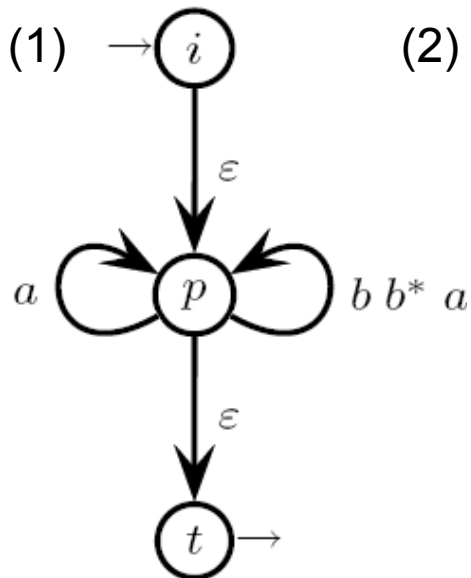


automaton



unique initial
and final nodes
without recirculation

normalized
automaton



EXERCISE:
eliminate the
nodes in the
order p, q

ELIMINATION OF INDETERMINISM – CONSTRUCTIVE PROCEDURE

For reasons of efficiency, usually the final version of a finite automaton ought to be in deterministic form

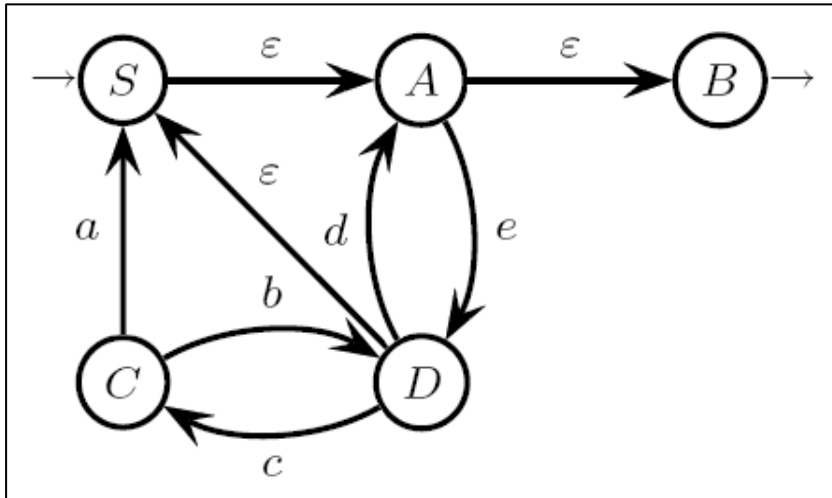
Every non-deterministic finite automaton can always be transformed into an equivalent deterministic one. Consequently every right linear grammar always admits an equivalent non-ambiguous right linear one. Thus every ambiguous regular expression can always be transformed into a non-ambiguous one

The algorithm to transform a non-deterministic automaton into a deterministic one is structured in two phases

1. Elimination of the spontaneous moves. As such moves correspond to copy rules, it suffices to apply the algorithm for removing the copy rules
2. Replacement of the non-deterministic multiple transitions by changing the automaton state set. This is the well known *subset construction*

Phase (1) can also be executed in many other ways. A practically convenient one is to cut off the ε -arcs by directly working on the state-transition graph

EXAMPLE – eliminating the ε -arcs by removing the copy rules from the grammar



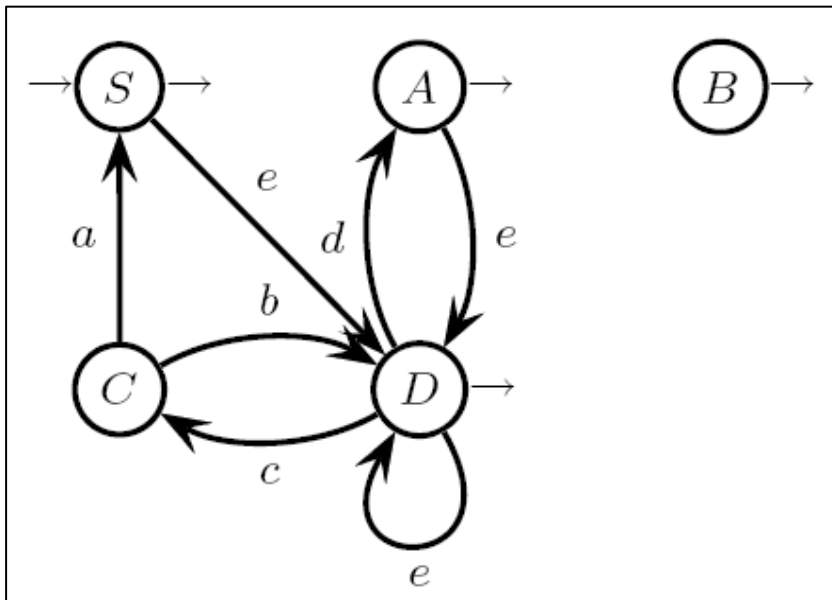
$$\left\{ \begin{array}{ll} S \rightarrow A & A \rightarrow B \mid e D \\ C \rightarrow a S \mid b D & D \rightarrow S \mid c C \mid d A \end{array} \right. \quad B \rightarrow \varepsilon$$

grammar with copy rules

	copy
S	S, A, B
A	A, B
B	B
C	C
D	D, S, A, B

unreachable

~~$B \rightarrow \varepsilon$~~



$$\left\{ \begin{array}{ll} S \rightarrow \varepsilon \mid e D & A \rightarrow \varepsilon \mid e D \\ C \rightarrow a S \mid b D & D \rightarrow \varepsilon \mid e D \mid c C \mid d A \end{array} \right.$$

grammar without copy rules

If after eliminating all the ε -arc the automaton is still non-deterministic, then go to the second phase

THE SUBSET CONSTRUCTION

Given the automaton N , non-deterministic and without spontaneous moves, construct the equivalent deterministic automaton M' as follows

$$p \xrightarrow{a} p_1 \qquad p \xrightarrow{a} p_2 \qquad \dots \qquad p \xrightarrow{a} p_k$$

If N contains $k \geq 1$ a -moves as above, then

- insert a new group state into M' to express the uncertainty among the possible k states $\{ p_1, p_2, \dots, p_k \}$
- insert the transitions outgoing from the new group state

$$p_1 \xrightarrow{a} \{ q_1, q_2, \dots \} \qquad p_2 \xrightarrow{a} \{ r_1, r_2, \dots \} \qquad \text{etc}$$

$$\{ q_1, q_2, \dots \} \cup \{ r_1, r_2, \dots \} \cup \dots$$

$$\{ p_1, p_2, \dots, p_k \} \xrightarrow{a} \{ q_1, q_2, \dots, r_1, r_2, \dots, \dots \}$$

Go on inserting new group states whenever necessary

THE SUBSET CONSTRUCTION

The deterministic automaton M' equivalent to the non-deterministic one N has

1. state set $Q' = \wp(Q)$, the power set of Q (set of subsets of Q)
2. final states F' , the subsets of Q that contain a final state of N
3. initial state $\{ q_0 \}$, a singleton
4. transition function δ' , see below

$$F' = \{ p' \in Q' \mid p' \cap F \neq \emptyset \}$$

$$\begin{array}{l} p' \in Q' \\ a \in \Sigma \end{array}$$

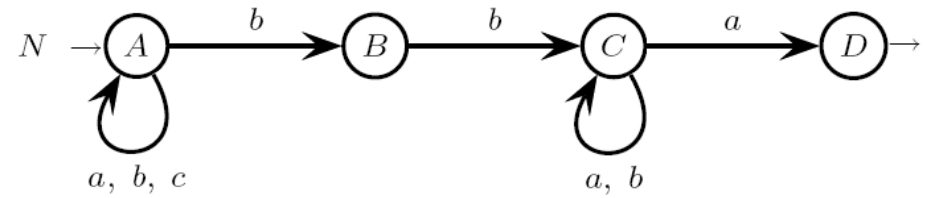
$$p' \xrightarrow{a} \left\{ s \mid q \in p' \wedge \text{arc } q \xrightarrow{a} s \text{ is in } \delta \right\}$$

NOTE

1. if state q goes to state q_{err} by arc a , then it is not necessary to add the error state to the group state, as the computations that go into the error state do not recognize any string and thus can be simply ignored
2. the elements (states) of set Q' are subsets of set Q , hence in the worst case the cardinality of Q' is an exponential function of that of Q , i.e., $\max 2^{|Q|}$, thus in general the deterministic automaton is larger than the non-deterministic one
3. automaton M' often has a few states that are unreachable from the initial one, hence they are useless; it may be better to include only the reachable states

EXAMPLE

start



non-deterministic automaton

$$\delta(A, b) = \{A, B\} \text{ create } \{A, B\}$$

$$\{A, B\} \xrightarrow{a} (\delta(A, a) \cup \delta(B, a)) = \{A\}$$

$$\{A, B\} \xrightarrow{b} (\delta(A, b) \cup \delta(B, b)) = \{A, B, C\}$$

final state

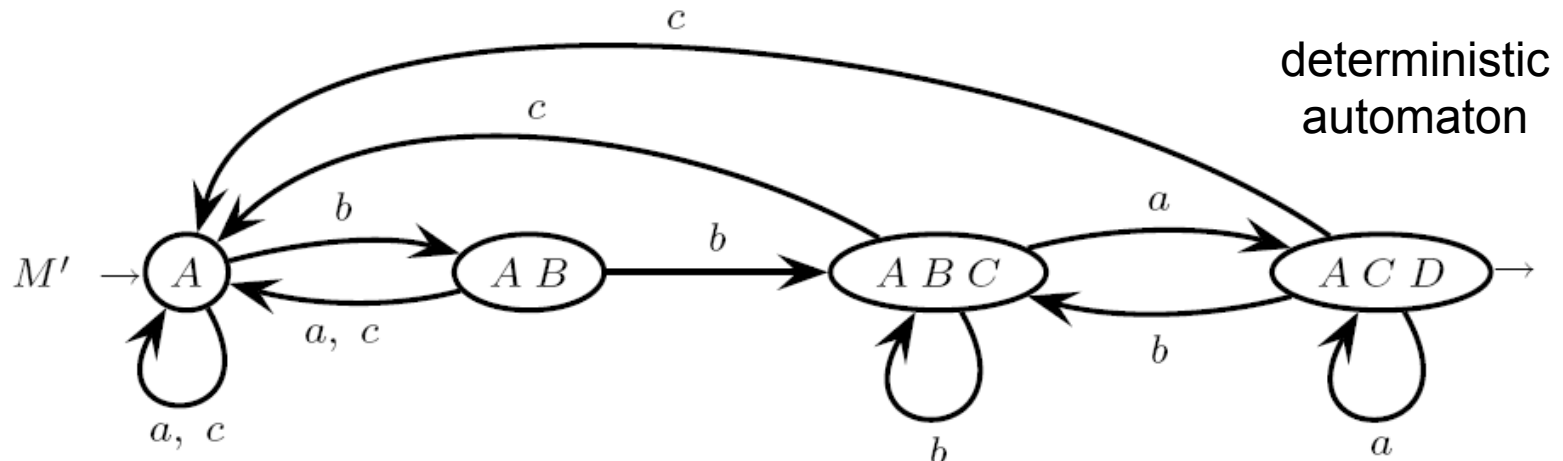
$$\{A, B, C\} \xrightarrow{a} (\delta(A, a) \cup \delta(B, a) \cup \delta(C, a)) = \{A, C, D\}$$

$$\{A, B, C\} \xrightarrow{b} (\delta(A, b) \cup \delta(B, b) \cup \delta(C, b)) = \{A, B, C\} \quad \text{end}$$

self-loop

Note: some subsets of Q may be unreachable, e.g., {A, C}

From the fourth step on no more group states are created



deterministic automaton

THE SUBSET CONSTRUCTION WORKS CORRECTLY. In fact a string x is accepted by automaton M' if and only if it is accepted by automaton N

- A) If a computation of N accepts x , then there is a path labeled by x from the initial state q_0 to a final state q_f . The algorithm makes sure that in M' there is one path labeled by $\{ q_0 \}$ to a group state $\{ \dots, q_f, \dots \}$ that contains q_f
- B) If x labels a valid computation of M' from $\{ q_0 \}$ to a final state $p \in F'$, then by construction p contains at least one final state q_f of M . Thus in M there is a path labeled by x from q_0 to q_f

PROPERTY – Every finite language is recognized by a deterministic finite state automaton. Therefore recognition can be carried out in real-time

COROLLARY – Every language recognized by a finite state automaton is generated by an unambiguous right linear (or left linear) grammar in correspondence with the automaton. Given a regular language presented in an ambiguous way, it is always possible to eliminate ambiguity and to present the same language unambiguously

FROM A REGULAR EXPRESSION TO A FINITE STATE AUTOMATON

There are a few algorithms to transform a regexp into an automaton, which differ as for automaton characteristic, e.g., det. vs non-det., automaton size and construction complexity

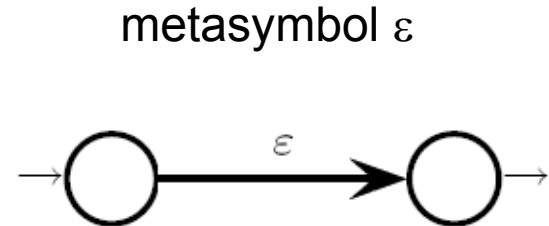
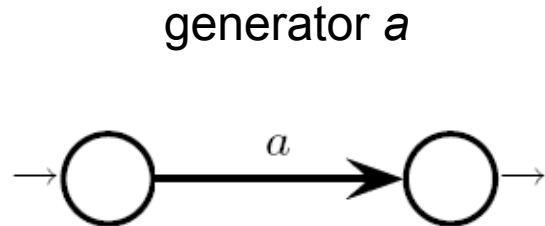
IN THE FOLLOWING THREE METHODS ARE PRESENTED

- 1) THOMPSON (or structural method)
 - 1) decomposes the regexp into subexpressions, until it reaches the atomic constituents thereof, i.e., the terminal symbols
 - 2) constructs the subexpression recognizers, connects them and builds up a network of recognizers that implement the union, concatenation and star operators
- 2) GLUSHKOV, MCNAUGHTON, YAMADA (GMY): constructs a non-det. recognizer without spontaneous moves, but with multiple transitions
- 3) BERRY-SETHI METHOD (BS): constructs a det. recognizer without spontaneous moves, but of size often larger than Thompson

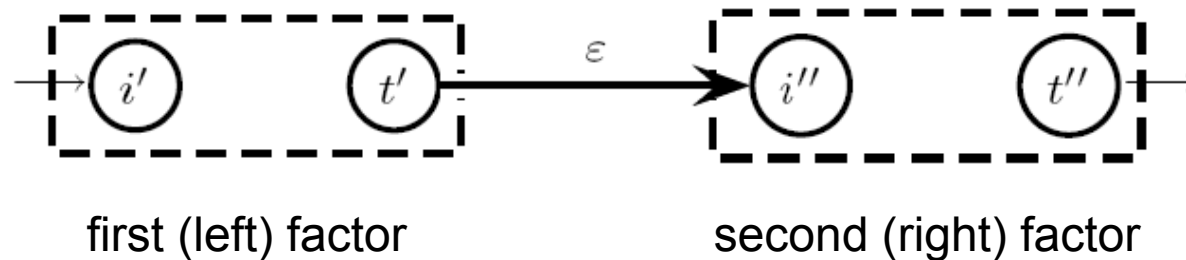
THOMPSON OR STRUCTURAL METHOD

- 1) modifies the original automaton to have unique initial and final states
- 2) is based on the correspondence of regexp and recognizer automaton

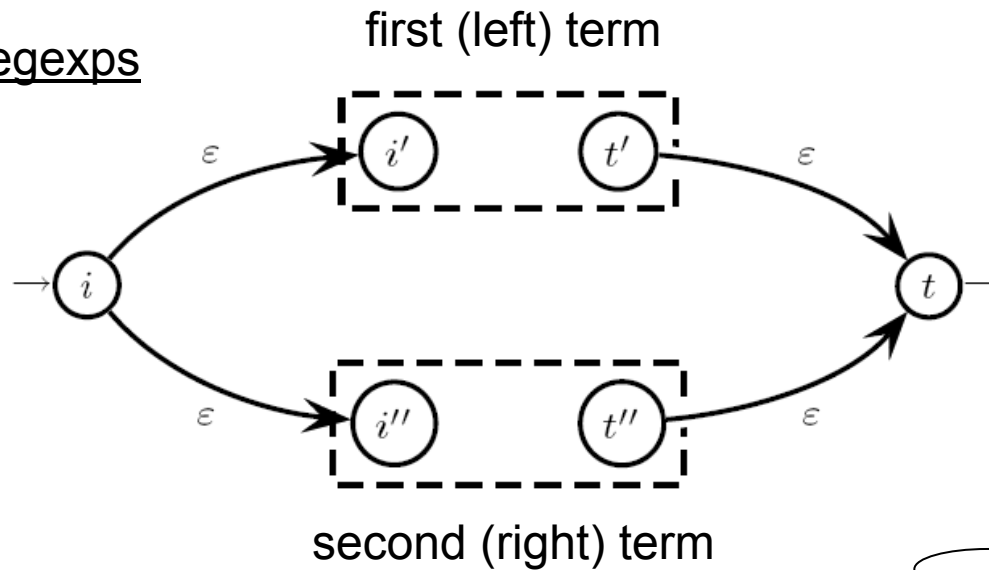
Recognizers of atomic regexps



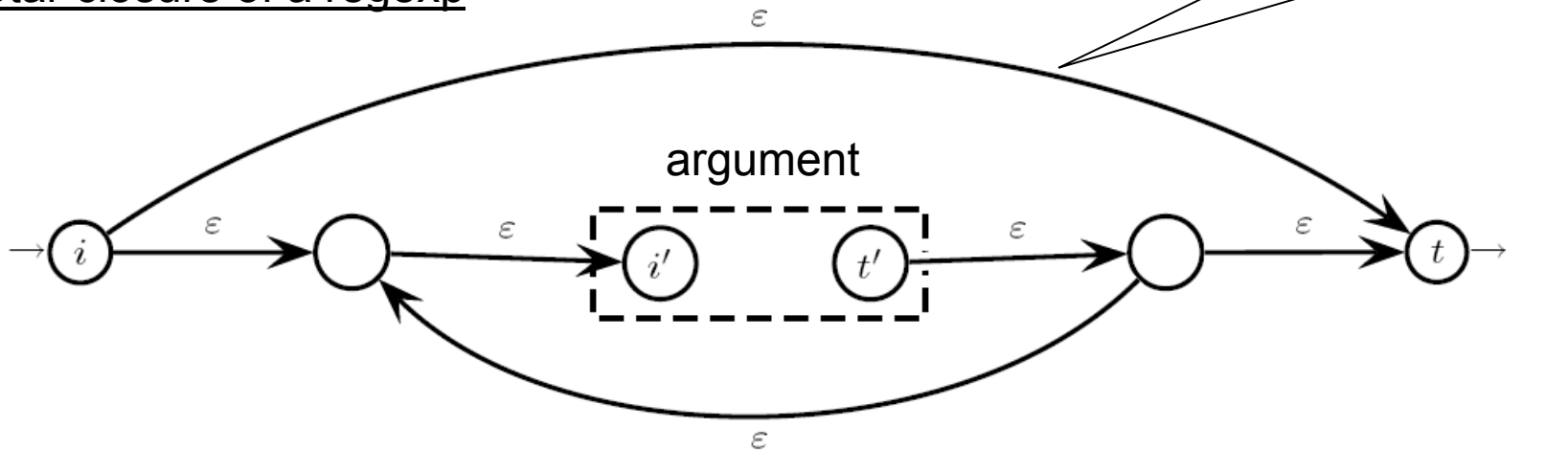
Concatenation of two regexps



Union of two regexps



Star closure of a regexp

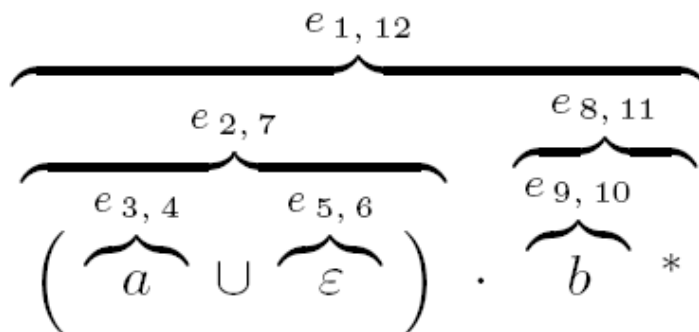


In general the outcome of the Thompson method is a non-deterministic automaton with spontaneous moves. The method is an application of the closure properties of the regular languages under the operations of union, concatenation and star

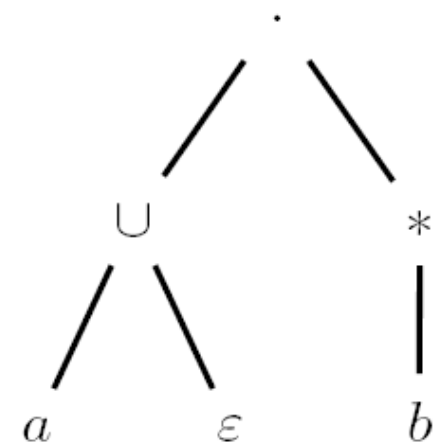
EXAMPLE $(a \cup \varepsilon) b^*$

parenthetization and
symbol / subexpression indexing

$$\left({}_1 \left({}_2 \left({}_3 a \right)_4 \cup \left({}_5 \varepsilon \right)_6 \right)_7 \cdot \left({}_8 \left({}_9 b \right)_{10} \right)^*_{11} \right)_{12}$$



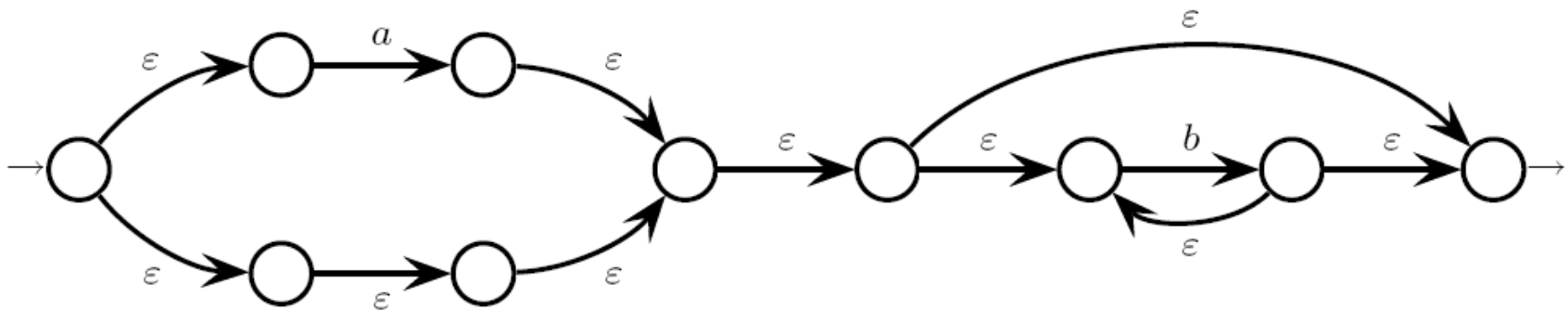
subexpression naming



structure tree

EXAMPLE (continued) $(a \cup \varepsilon) b^*$

Thompson non-deterministic automaton of the regexp,
with spontaneous moves



There are various optimizations of the Thompson method that avoid to create redundant states, e.g., the initial and final ones, or that avoid the spontaneous moves at an early stage of the construction