Non-Deterministic Automata and Language Recognition

Prof. Licia Sbattella

aa 2007-08

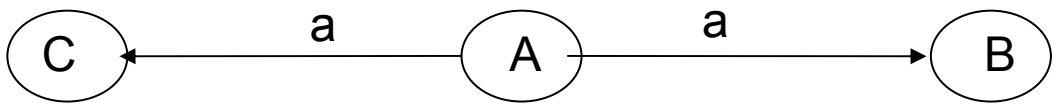
Translated and adapted by L. Breveglieri

NON-DETERMINISTIC AUTOMATA (or INDETERMINISTIC)

A right linear grammar may well contain the following alternative, that gives rise to a non-deterministic behaviour:

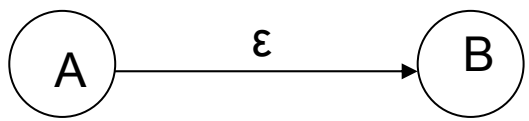
$$A \to aB \mid aC \quad \text{where} \quad a \in \Sigma$$

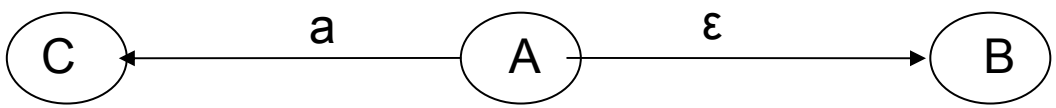
$$\delta(A, a) = \{B, C\}$$



Otherwise the non-deterministic behaviour may originate from the presence of a spontaneous move (or ε move), which can take place without reading any input symbol.

$$A \to B \quad \text{where} \quad B \in V$$



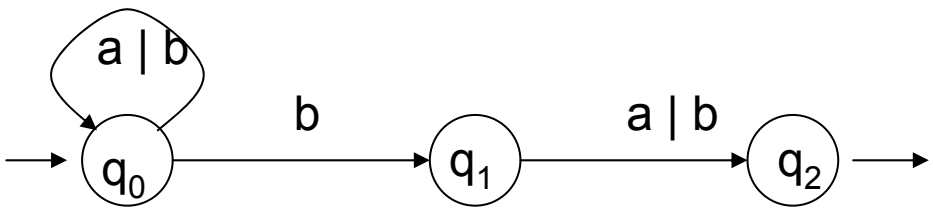


MOTIVATION OF INDETERMINISM

1) The correspondence between grammars and automata leads to introduce the idea of transitions with two or more destination states and of spontaneous moves (the two main sources of non-determinism), and possibly also of two or more initial states (the third possible source of non-determinism), which corresponds to a grammar with two or more axioms.
2) Concision: the definition of a language by means of a non-deterministic machine may be more readable and compact.

EXAMPLE – (second last char. = b)

$$L_2 = (a \mid b)^* b (a \mid b)$$



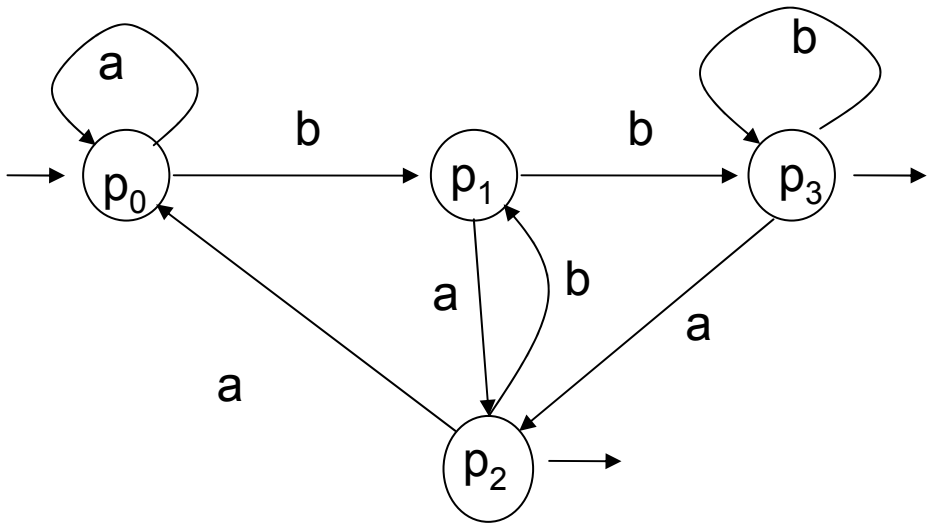
Recognition of *baba. Recognized by:*

$$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2$$

Not recognized by other possible computations (because a final state is not reached):

$$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0$$

The same language is accepted by the non deterministic M2, which does not make as evident that the second last character of the accepted strings has to be *b*.



EXERCISE

If the example is generalized, in passing from the language L_2 to the language L_k such that the *k*-th last element ($k \geq 2$) is *b,* one sees that the non-det. automaton has $k + 1$ states, while it can be proved that the minimum number of states of a det. automaton that recognizes the same language must be an exp. function of *k*.

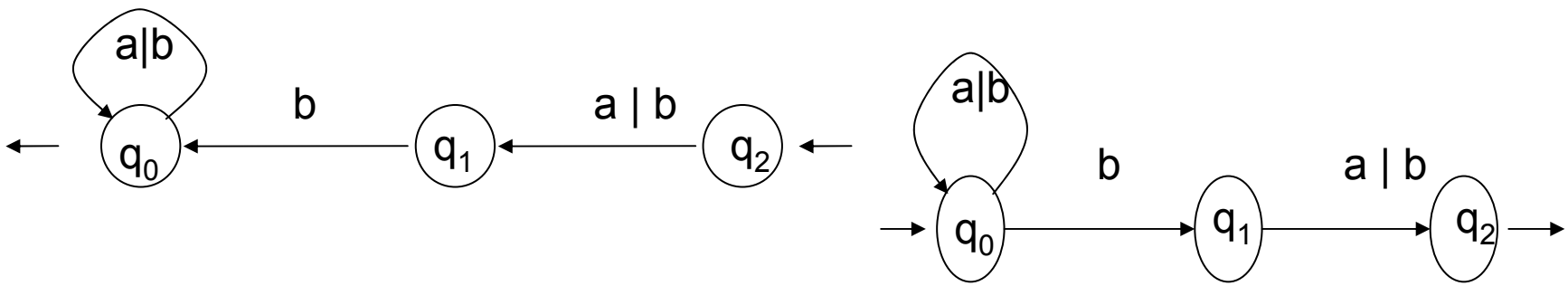
Allowing non-determinism may make some language definitions more coincise.

3) Left-to-right duality.

Another situation leading 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 pairs of converging moves with the same label, or by swapping initial and final states thus having two or more initial states.

EXAMPLE - The language of all the strings with second last char. equal to *b* is the mirror image of that of the strings with second char. equal to *b*.

$$L' = \{x \mid b \quad \text{is the second character of} \quad x\} \quad L_2 = (L')^R$$



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

NON-DETERMINISTIC RECOGNITION

For the moment ignore the existence of spontaneous moves. Here follows the formal definition of non-deterministic computation:

A *non-deterministic finite state automaton N* without spontaneous moves consists of:

1. a set fo states Q
2. an input (or terminaL) alphabet Σ
3. two subsets of Q: the sets I and F of the initial and final states, respectively
4. the set δ of the transitions, which is a subset of the cartesian product $Q \times \Sigma \times Q$

A computation that originates at q_0 and ends at q_n, of length *n*, with label $a_1a_2...a_n$

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots \xrightarrow{a_n} q_n$$

$$q_0 \xrightarrow{a_1 a_2 \ldots a_n} q_n$$

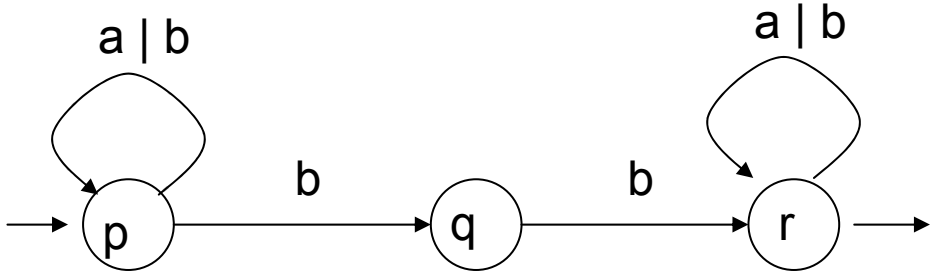
An input string *x* is accepted (recognized) by the automaton if it is the labeling of a path that starts from an initial state and ends into a final state.

Language recognized by the automaton N:

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

EXAMPLE – text search for a word

Given a word *y*, to recognize whether a text fragment contains that word submit the text to the finite automaton accepting the language $(a \mid b)^* y (a \mid b)^*$. Suppose for instance $y = bb$.



The string *abbb* is the label of several computations starting from the initial state.

$$p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{b} p \xrightarrow{b} p \qquad p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{b} p \xrightarrow{b} q$$

$$p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{b} q \xrightarrow{b} r \qquad p \xrightarrow{a} p \xrightarrow{b} q \xrightarrow{b} r \xrightarrow{b} r$$

The first two computations do not locate the searched word. The last two find it; in the two following positions, respectively

$$a\underbrace{bb}b \text{ and } ab\underbrace{bb}$$

TRANSITION FUNCTION

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

Given the non-det. automaton N = (Q, Σ, δ, I, F), without spontaneous moves, the transition function δ is defined as to have the following domain and image:

$$\delta : (Q \times (\Sigma \cup \varepsilon)) \rightarrow \text{ finite subsets of } Q$$

For a terminal character *a:*

$$\delta(q, a) = [p_1, p_2, ..., p_k]$$

$$\forall q \in Q : \delta(q, \varepsilon) = [q]$$

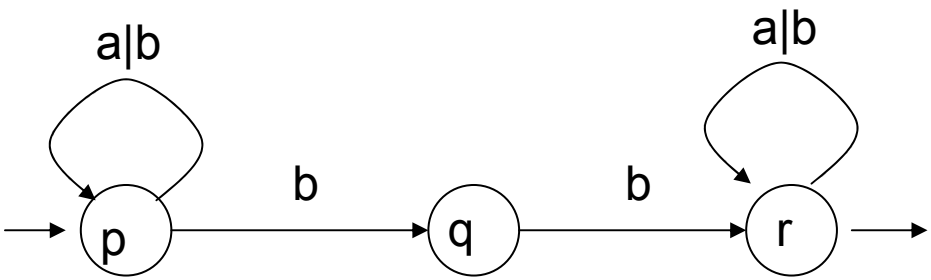
For any terminal string *y:*

$$\forall q \in Q, y \in \Sigma^* : \delta(q, y) = [p \mid q \overset{y}{\rightarrow} p]$$

Here is the language accepted by N: the string *x* is accepted if after scanning it the current set of states contains at least one final state.

$$L(N) = \left\{ x \in \Sigma^* \mid \exists q \in I : \delta(q, x) \cap F \neq \varnothing \right\}$$

EXAMPLE – text search for a word (continued)



$$\delta(p,a)=[p],$$
$$\delta(p,ab)=[p,q],$$
$$\delta(p,abb)=[p,q,r]$$

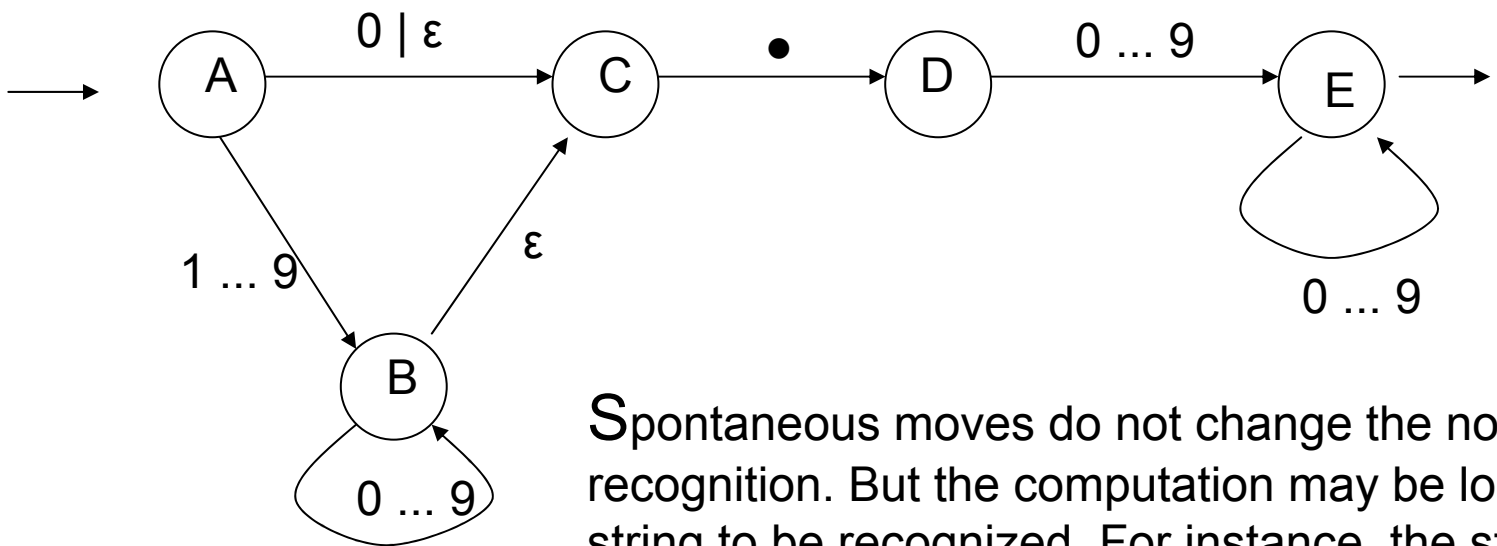
AUTOMATA WITH SPONTANEOUS MOVES (ε-MOVES)

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

Using ε -arcs makes easy the modular construction of finite automata, and thus allows to reuse other automata or fragments thereof.

EXAMPLE – decimal constant (where union and concatenation are used)

$$L = (0 \mid \varepsilon \mid N)\,\square\,(0...9)^{+} \quad \text{where} \quad N = (1...9)(0...9)^{*}$$



Spontaneous moves do not change the notion of string recognition. But the computation may be longer than the string to be recognized. For instance, the string "34●5" is accepted by the following computation (of 5 steps):

$$A \xrightarrow{3} B \xrightarrow{4} B \xrightarrow{\varepsilon} C \xrightarrow{\bullet} D \xrightarrow{5} E$$

UNIQUENESS OF THE INITIAL STATE. A non-det. automaton may well have two or more initial states. However, it is easy to construct an equivalent non-det. aut. with only one initial state. Add a new initial state q_0, connect it to the existing initial states with as many ε-moves, unlabel them as initial and leave only q_0.

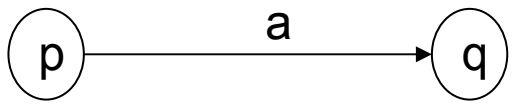
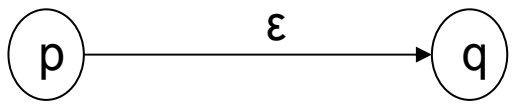
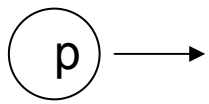
One sees soon that a computation of the new automaton accepts a string if and only if the old automaton accepts it as well. The additional ε-moves can be eliminated, as it will be shown in the following.

CORRESPONDENCE BETWEEN AUTOMATON AND GRAMMAR

Suppose that:

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

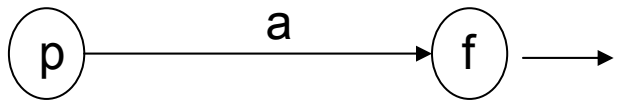
$N = (Q, \Sigma, \delta, q_0, F)$ is the automaton (assume the initial state is unique).

Right linear grammar	*Finite automaton*
1. non-terminal alphabet V	state set $Q = V$
2. axiom $S = q_0$	initial state $q_0 = S$
3. $p \to aq,\ a \in \Sigma \quad p, q \in V$	
4. $p \to q$ where $p, q \in V$	
5. $p \to \varepsilon$	final state 

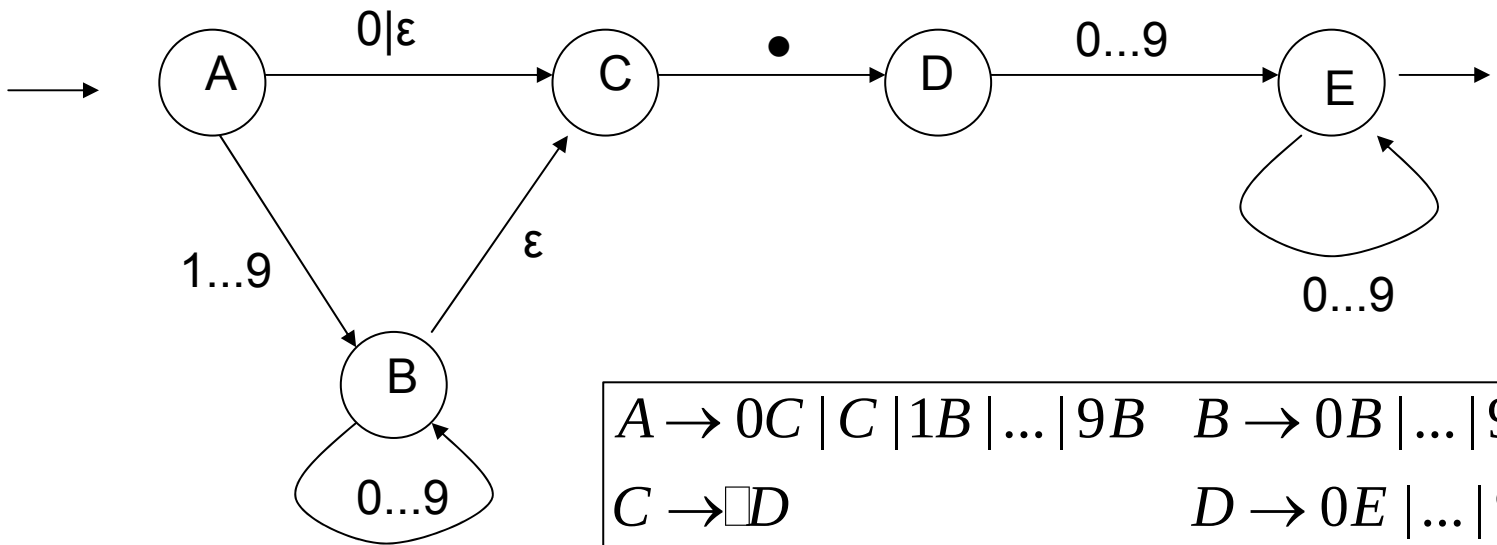
A grammar derivation corresponds to an automaton computation, and viceversa. Consequently the two models define the same language.

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

If the (right linear) grammar contains also terminal rules of the type $p \rightarrow a$ with $a \in \Sigma$, the automaton contains also a final state *f*, in addition to those corresponding to ε-productions of the grammar, and the following move:



EXAMPLE – equivalence between right linear grammars and automata

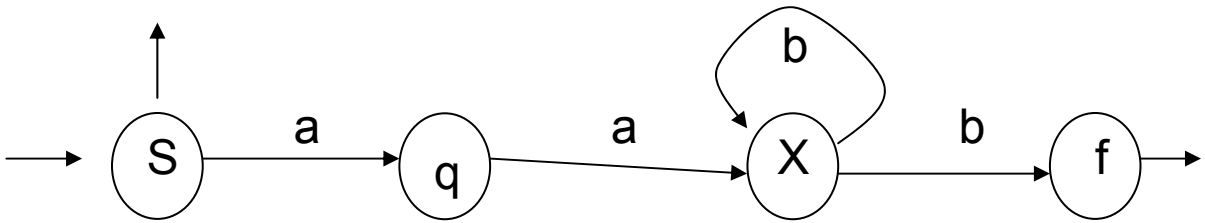


$A \rightarrow 0C \mid C \mid 1B \mid ... \mid 9B \quad B \rightarrow 0B \mid ... \mid 9B \mid C$

$C \rightarrow \square D \qquad D \rightarrow 0E \mid ... \mid 9E$

$E \rightarrow 0E \mid ... \mid 9E \mid \varepsilon$ where A is the axiom

EXAMPLE – right linear grammar (but not strictly linear)

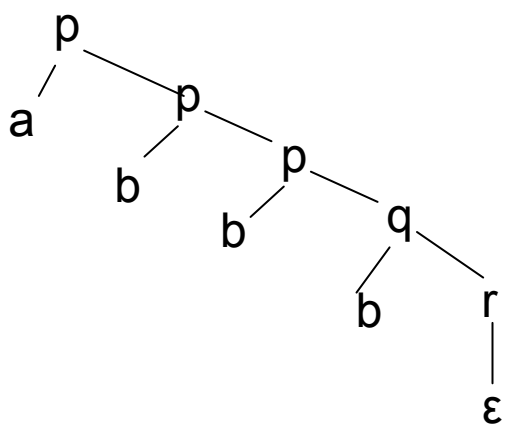


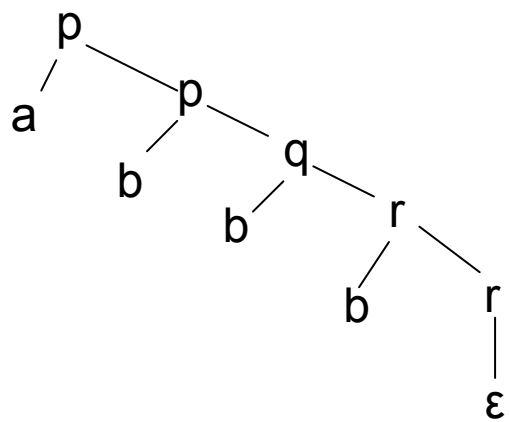
$$S \to aaX \mid \varepsilon$$
$$X \to bX \mid b$$

GRAMMAR AMBIGUITY – As grammar derivations are in one-to-one correspondence with automaton computations, the notion of ambiguity can be extended to automata: an automaton is ambiguous if and only if the corresponding grammar is so. In practice, the automaton is ambiguous if a string *x* labels two (or more) different accepting paths.

EXAMPLE – search a text for a word – recognize *abbb*

$$p \to ap \mid bp \mid bq$$
$$r \to ar \mid br \mid \varepsilon$$
$$q \to br$$





LEFT LINEAR GRAMMAR AND AUTOMATON

$$A \to Ba \quad A \to B \quad A \to \varepsilon$$

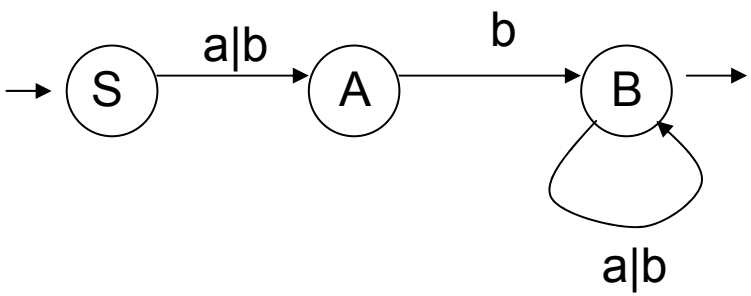
$$L^R = (L(G))^R \quad \text{generated by} \quad G_R$$

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

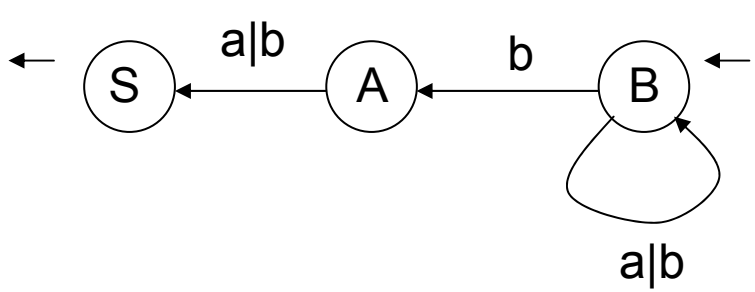
$$G: \; S \to Aa \mid Ab \quad A \to Bb \quad B \to Ba \mid Bb \mid \varepsilon$$

$$G_R: \; S \to aA \mid bA \quad A \to bB \quad B \to aB \mid bB \mid \varepsilon$$

recognizer of $(L(G))^R$:



recognizer of L(G):



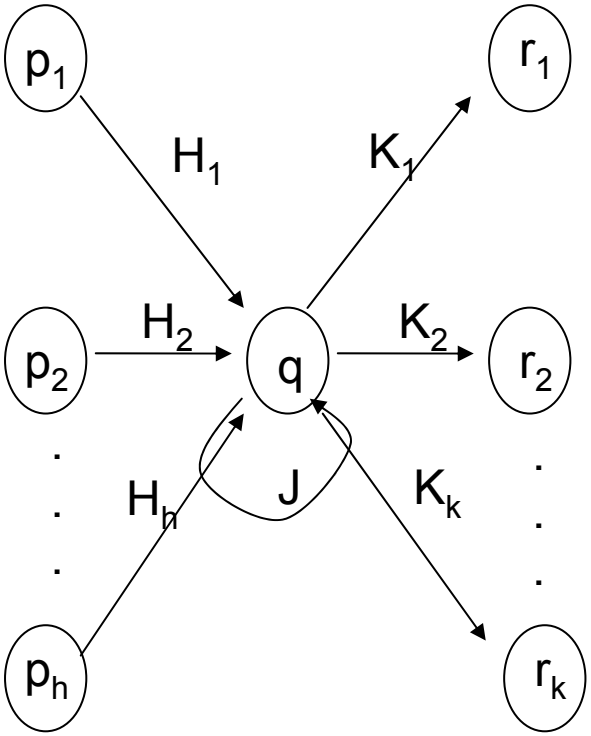
FROM THE AUTOMATON TO THE REGEXP DIRECTLY
BMC METHOD (Brzozowski and McCluskey)
ALSO KNOWN AS “NODE ELIMINATION” METHOD

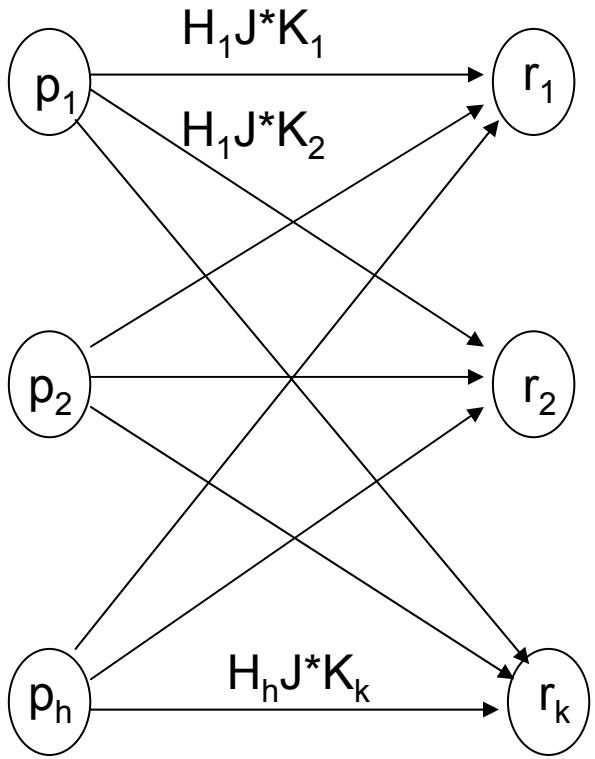
Assume the initial and final states *i* and *t* are unique and do not have incoming or outgoing arcs, respectively. If it is not so, modify the automaton, as explained before, to have both unique.

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

Construct the so-called GENERALIZED FINITE AUTOMATON: it is defined as the normal one, but the arcs can be labeled by regular expressions, not only individual characters of the terminal alphabet.

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, and only one arc going from *i* to *t*. The regexp labeling such an arc generates the complete language recognized by the original finite automaton.



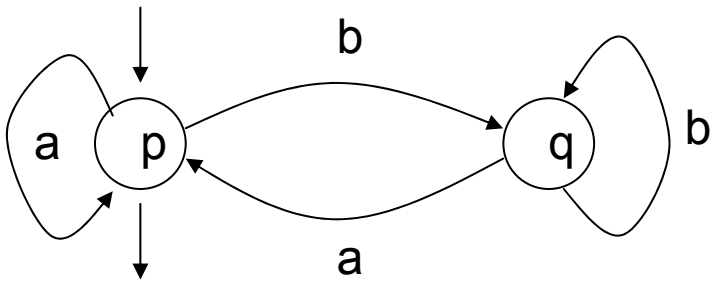


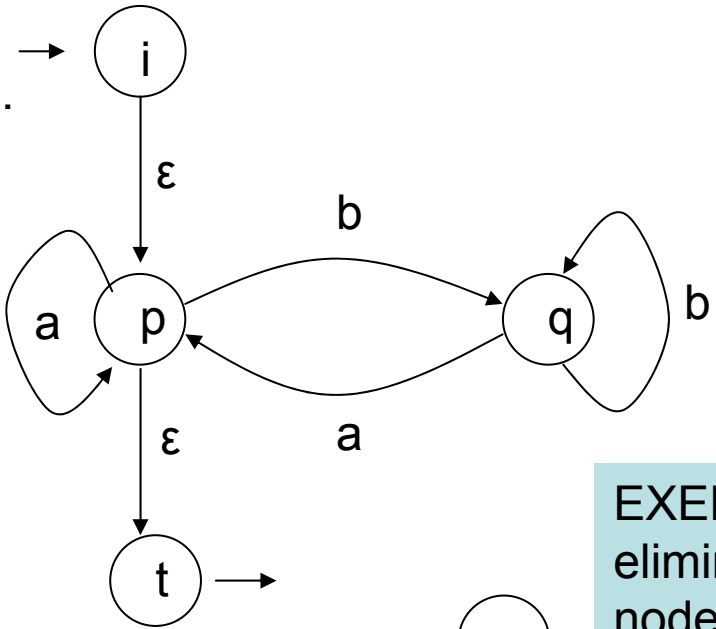
For every state pair p_i, r_j, there is the arc:
(sometimes p_i and r_j may coincide (self-loop))

$$p_i \xrightarrow{H_i J^* K_j} r_j$$

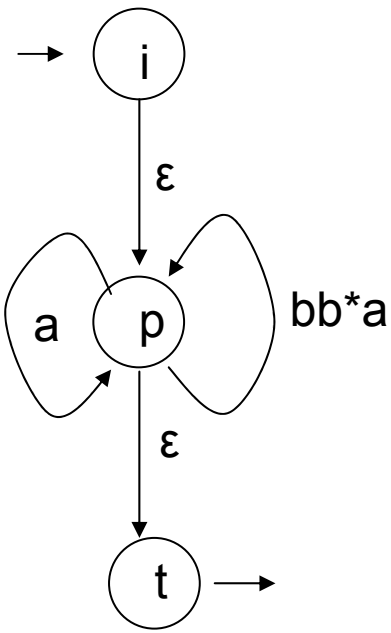
The elimination order is not relevant. However, different orders may generate different regexps, all equivalent but of different complexity.

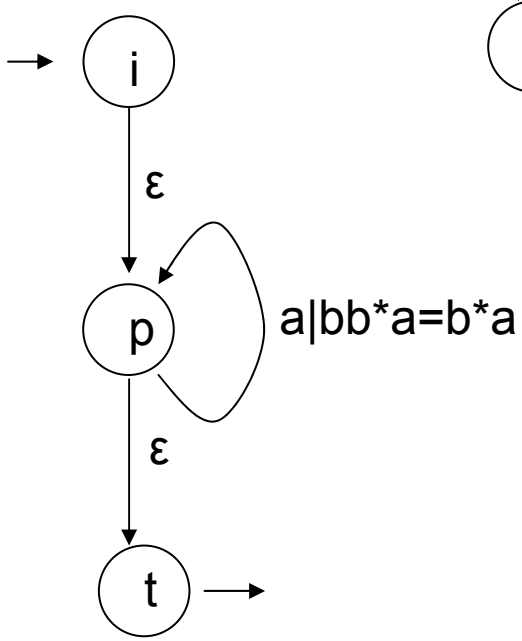
EXMPLE – normalization of the aut.
and then node elimination (order: *q*, *p*).

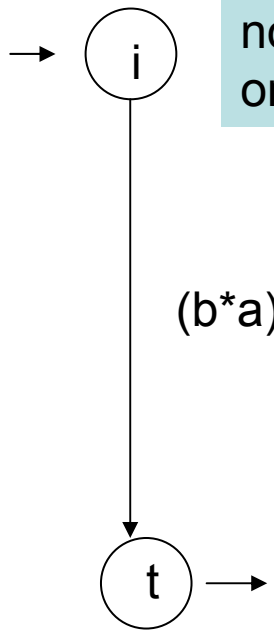




EXERCISE:
eliminate the nodes in the order: *p*, *q*.







ELIMINATION OF INDETEMINISM – CONSTRUCTIVE PROCEDURE

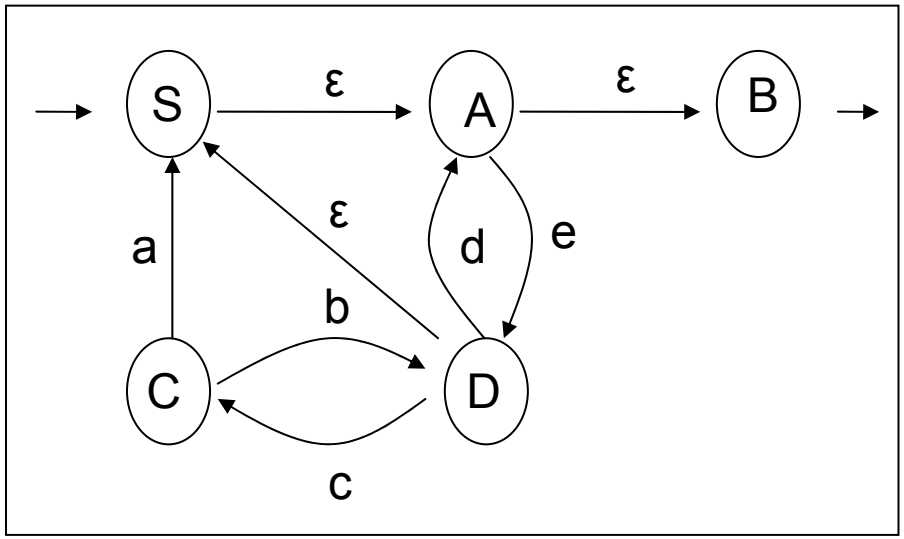
For reasons of efficiency, usually the final version of a finite automaton should be in deterministic form.

Every non-deterministic finite automaton can always be transformed into an equivalent deterministic one, and consequently every right linear grammar always admits an equivalent non-amboguous right linear one, and therefore every ambioguous regexp can always be transformed into a non-ambiguous one.

The algorithm to transform a non-det. automaton into a det. one is structured in two phases:

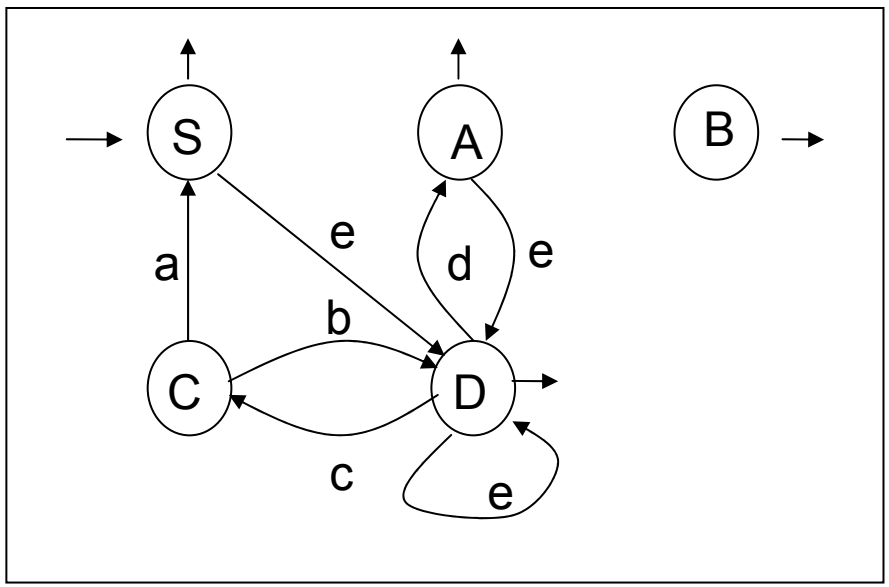
1. ELIMINATION OF SPONTANEOUS MOVES (as such moves correspond to copy rules, it suffices to apply the algorithm for removing copy rules).
2. REPLACEMENT OF NON-DET. TRANSITIONS by means of one (this is the celebrated SUBSET CONSTRUCTION)

EXAMPLE



$$S \to A \qquad A \to B \mid eD \qquad B \to \varepsilon$$
$$C \to aS \mid bD \qquad D \to S \mid cC \mid dA$$

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



$$S \to \varepsilon \mid eD \qquad A \to \varepsilon \mid eD \qquad B \to \varepsilon$$
$$C \to aS \mid bD \qquad D \to \varepsilon \mid eD \mid cC \mid dA$$

If, after eliminating all the spontaneous moves, the automaton is still non-det., go to the second phase.

THE SUBSET CONSTRUCTION

Given the automaton N, non-det. and withoput spontaneous moves, construct the equivalent deterministic automaton M', as follows

If N contains the *k* moves aside:

$$p \xrightarrow{a} p_1, \quad p \xrightarrow{a} p_2, \quad \ldots, \quad p \xrightarrow{a} p_k,$$

$$[p_1, p_2, \ldots, p_k]$$

- insert into M' a new group state, with the function of expressing the uncertainty among the possible *k* states
- then insert the transitions outgoing from the new group state:

$$p_1 \xrightarrow{a} [q_1, q_2, \ldots], \quad p_2 \xrightarrow{a} [r_1, r_2, \ldots], \quad \text{etc.}$$

$$[q_1, q_2, \ldots] \cup [r_1, r_2, \ldots] \cup \ldots$$

$$[p_1, p_2, \ldots, p_k] \xrightarrow{a} [q_1, q_2, \ldots, r_1, r_2, \ldots, \ldots]$$

and go on inserting new group states whenever necessary

THE SUBSET CONSTRUCTION

The deterministic automaton M' equivalent to N has:

1. state set Q' = P (Q), the power set of Q
2. final states F', the subsets of Q that contain a final state of N

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

3. initial state $[q_0]$
4. transition function δ':

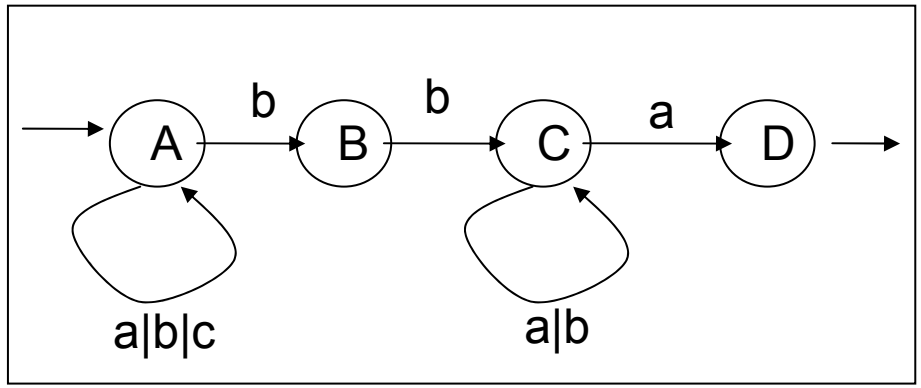
$$\forall p' \in Q',\ \forall a \in \Sigma$$

$$p' \xrightarrow{a} [s \mid q \in p' \wedge (q \xrightarrow{a} s \text{ is in } N)]$$

NOTE:

1. if q goes to q_{err} by the arc a, the error state need not be added to the group state, as the computations that go into the error state do not recognize any string and can be simply ignored
2. the elements (states) of Q' are the subsets of Q and hence the cardinality of Q' is, in the worst case, an exp function of that of Q (max $2^{|Q|}$, therefore the det. automaton is, in general, larger than the non-det. one)
3. often M' has some states that are unreachable from the initial one, and hence unuseful; it may be convenient to include from start only the reachable states

EXAMPLE



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

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

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

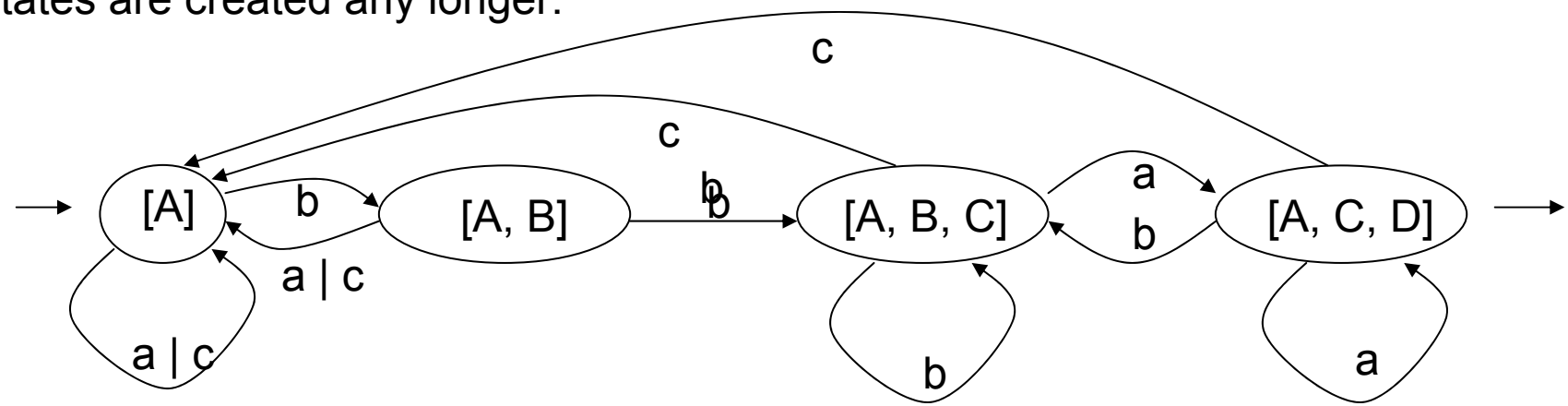
$[A,B,C] \xrightarrow{a} (\delta(A,a) \cup \delta(B,a) \cup \delta(C,a)) = [A,C,D]$

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

Note: not all the subsets of Q are reachable (e.g., [A, C] would be unuseful).

end

The algorithm terminates at step number 4, as from that step on no new group states are created any longer.



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

A) If a computation of N accepts x, there exists a path labeled by x from the initial state q_0 to a final state q_f. The algorithm ensures that in M' there has to exist (only) one path labeled by $[q_0]$ to a group state $[....,q_f,....]$ containing q_f.

B) If x is the label of a valid computation of M', from q_0 to some final state $p \in F'$, then by construction p has to contain at least one final state q_f of M. And then, again by construction, there has to exist a path labeled by x from q_0 to q_f.

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

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

FROM THE REGULAR EXPRESSION TO THE RECOGNIZER AUTOMATON

There are several algorithms, which differ as for the characteristic of the automaton (det. vs non-.det., etc), as for the size of the automaton and as for the complexity of the construction.

IN THE FOLLOWING TWO METHODS ARE PRESENTED:

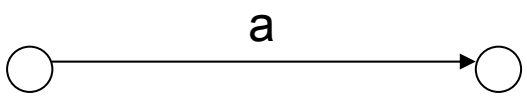
1) THOMPSON METHOD (or structural method):
    1) the regexp is decomposed into subexpressions, until the atomic constituents are reached (i.e. terminal symbols)
    2) constructs the recognizer of the various subexpressions, connects them and builds up a network of recognizers that implement the union, concatenation and star operators
2) GLUSHKOV, MCNAUGHTON AND YAMADA METHOD (GMY): directly constructs a deterministic recognizer, without spontaneous moves, but in general of size larger than the Thompson method does.

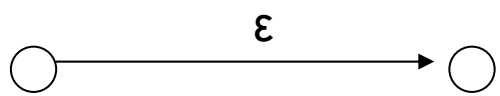
BOTH METHODS can be combined with the determinization algorithm.

THOMPSON OR STRUCTURAL METHOD

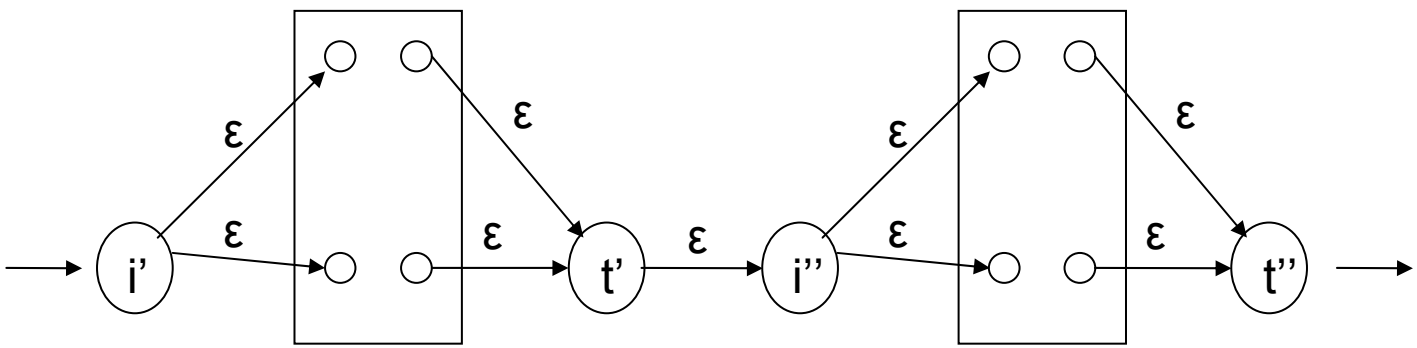
1) modifies the original automaton so as to have unique initial and final states
2) the method is based on the correspondence between regexp and rec. automaton

Recognizers of atomic regexps:

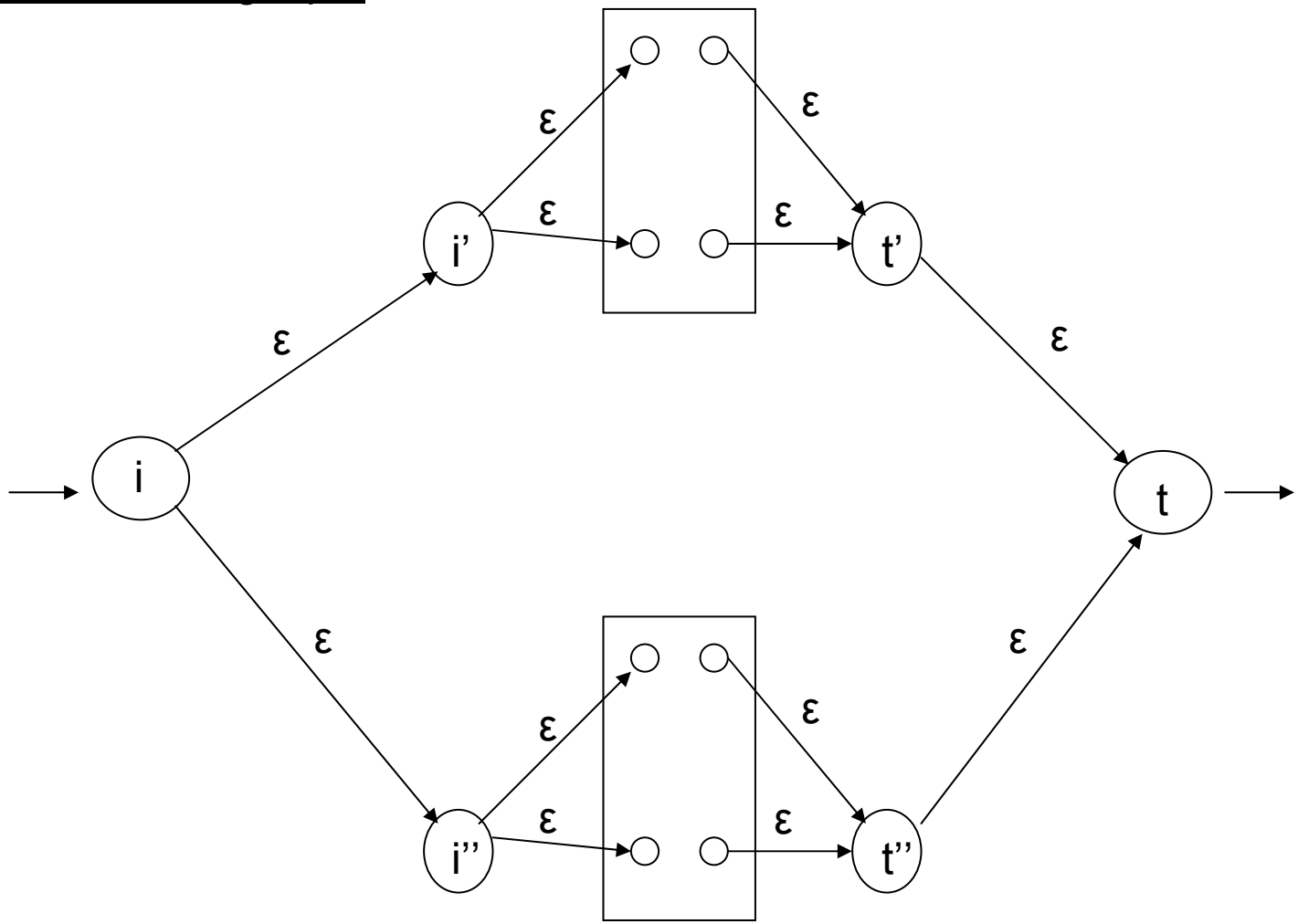




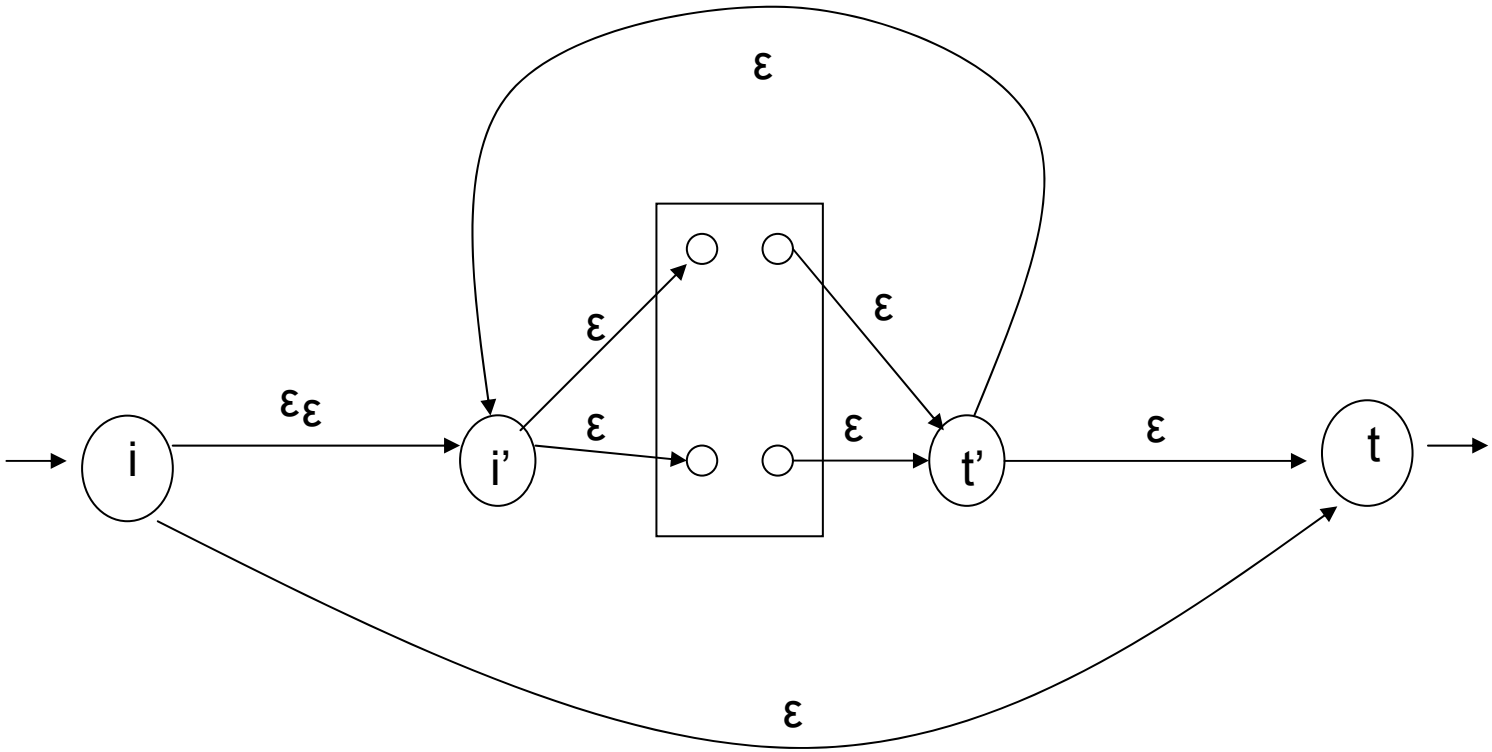
Concatenation of two regexps:



Union of two regexps:

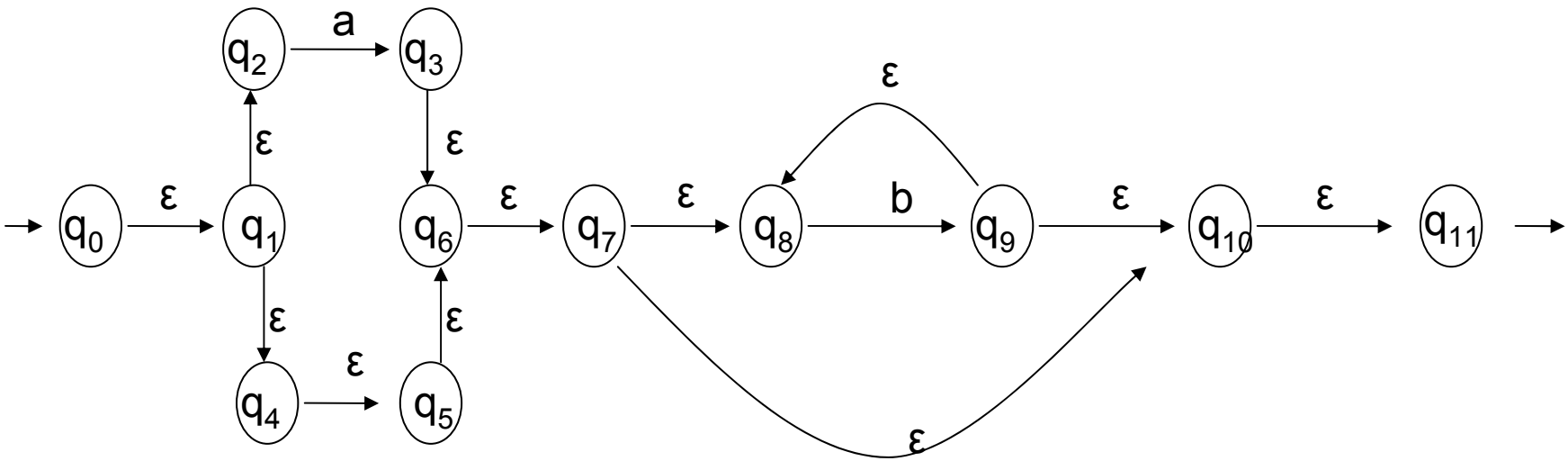


Star closure of a regexp:



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

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



There are improved versions of the Thompson method, that avoid the creation of redundant states (e.g., q_0 and q_{11}), or that eliminate soon spontaneous moves.

Bibliography

- S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006
- Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969
- A. Salomaa – *Formal Languages*, Academic Press, 1973
- D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987
- L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d'Esame Risolti,* web site (eng + ita)