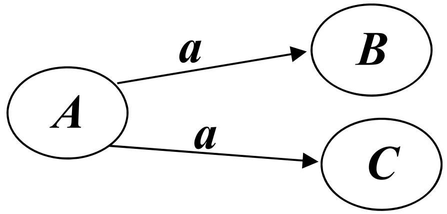
Nondeterministic Finite Automata

Prof. A. Morzenti

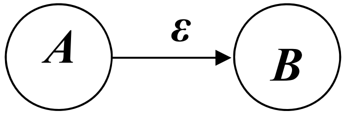
FORMS OF NONDETERMINISM

1) alternative moves for a unique input

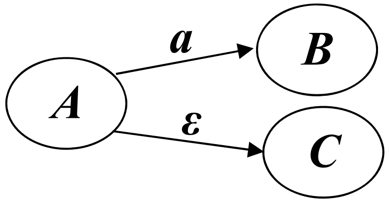


$\boxed{\delta(A,a)=\{B,C\}}$

2) spontaneous move (or ε-move): automaton changes its state without "consuming" input



$\delta(A,\varepsilon)=B$



$\delta(A,a)=B \quad \delta(A,\varepsilon)=C$

3) distinct initial states (useful, e.g., when merging various automata …)

ANALOGY WITH RIGHT-LINEAR GRAMMARS

1) grammar with two alternatives $A \to aB \mid aC$ with $a \in \Sigma$

2) Grammar with copy rule $A \to B$ with $B \in V$

3) grammar(s) with «many axioms» (useful when composing grammars …)

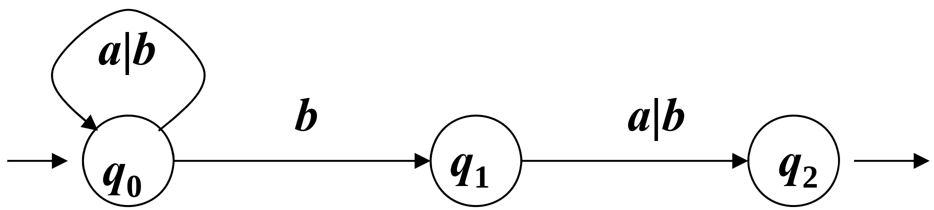
FOUR MOTIVATIONS FOR NONDETERMINISM IN FINITE STATE AUTOMATA

1) matching/mapping between grammars and automata
   already illustrated

2) conciseness: language definitions through ND automata are more compact and readable

Example – language with penultimate symbol = *b*

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

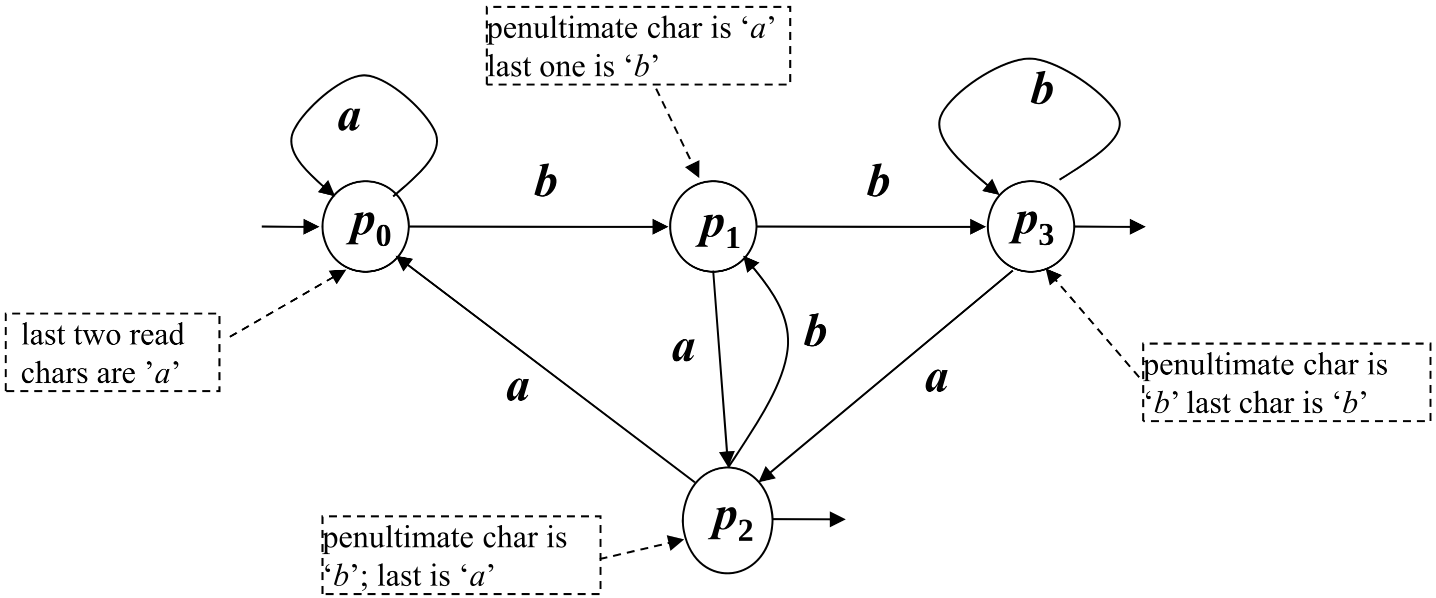


recognition of *baba*
two computations, only one accepting

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

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

the same language accepted by a ***deterministic*** automaton $M2$
in $M2$ the condition that the symbol before the last one is a b is not so clear



Exercise: show/prove that
Generalizing the example, from language L_2 to language L_k where the k-th last element ($k \geq 2$) is b, the ***nondeterministic*** automaton has ***k + 1 states***,
the number of states of the ***deterministic*** one grows ***exponentially*** with ***k*** **(≈2^k)**

nondeterminism can make certain definitions much more concise

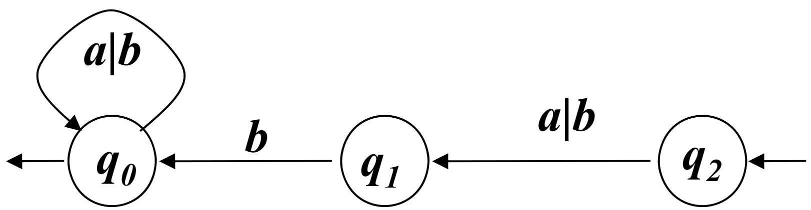
3) left – right duality
going from a (deterministic) automaton for lang. L to that for L^R requires to:

1. switch initial and final states
2. reverse the arrows direction

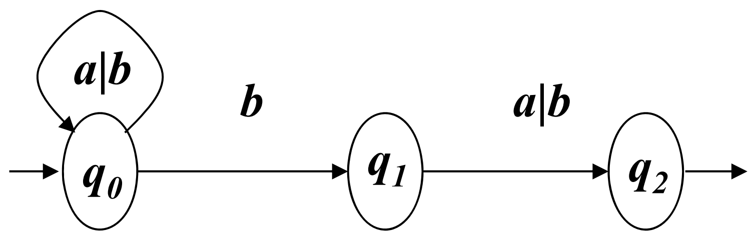
Both operations may introduce nondeterminism

Example - The language of strings having *b* as the penultimate symbol is the reverse image of the one having *b* as the second symbol

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



b second char: deterministic

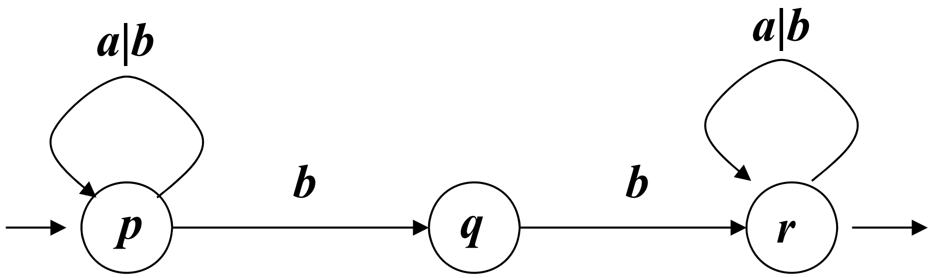


b penultimate char: nondeterministic

4) the transition through nondeterministic automata is useful in the construction of the finite recognizer of a language defined by a regular expression (see coming lessons)

Example – Search of a string in a text

Given a word $y \in \{a,b\}^*$, to check its inclusion in a text, we scan the text with the automaton accepting the language $\boldsymbol{(a \mid b)^* \, y \, (a \mid b)^*}$. Example with $y = bb$



$$\delta(p,a) = \{p\},\quad \delta(p,ab) = \{p,q\},\quad \delta(p,abb) = \{p,q,r\}$$

The text (ex. string ***abbb***) labels many computations from initial to final states

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

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

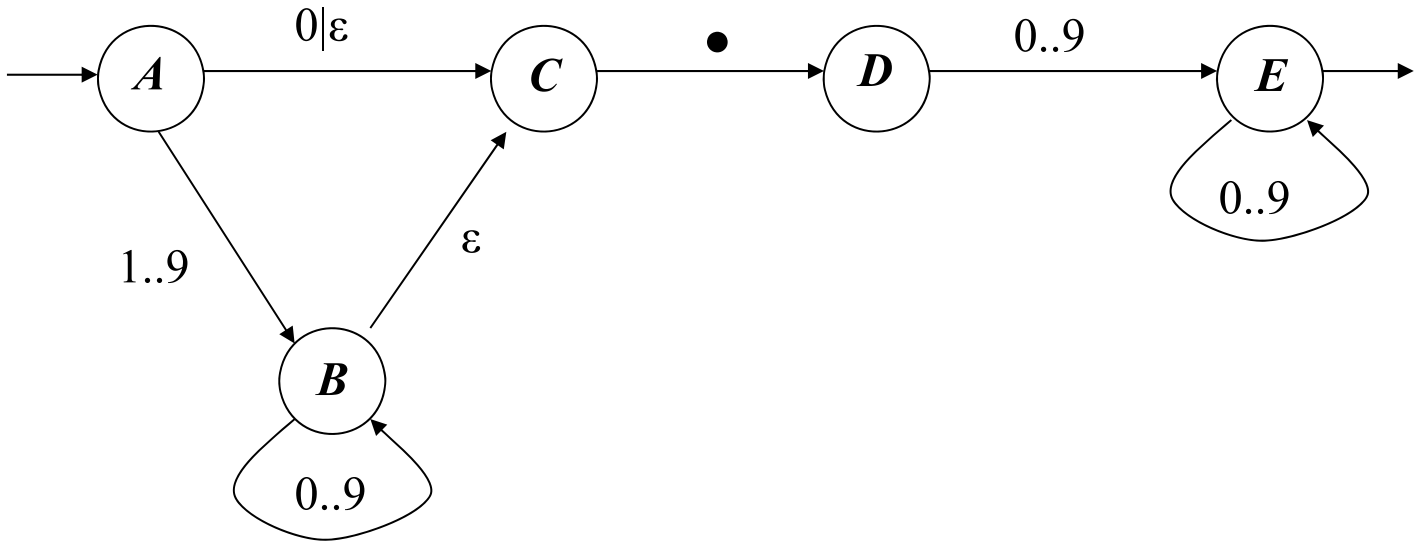
The first two computations do not «find» the word. The two last ones do

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

AUTOMATA WITH SPONTANEOUS MOVES

Example – decimal constants (with or without 0 before the dot, with no leading 0's in the integer part)

$$L = \left(0 \mid \varepsilon \mid \left((1..9)(0..9)^*\right)\right) \bullet (0..9)^+$$



When the automaton has spontaneous moves, the computation can be longer than the string (NB: the property of real-time analysis does not hold)

Computation accepting string 34•5: $A \xrightarrow{3} B \xrightarrow{4} B \xrightarrow{\varepsilon} C \xrightarrow{\bullet} D \xrightarrow{5} E$

Computation accepting string •02: $A \xrightarrow{\varepsilon} C \xrightarrow{\bullet} D \xrightarrow{0} E \xrightarrow{2} E$

UNIQUENESS OF THE INITIAL STATE

A nondeterministic automaton can have many initial states

But it is easy to obtain an equivalent one with a unique initial state …

… by adding a «new» initial state q_0 and the ε-moves from q_0 to the former initial states

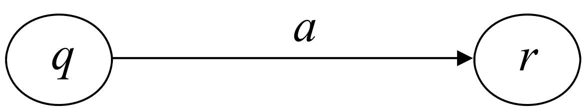
The added moves can then be eliminated (we will see how…)

EQUIVALENCE OF FINITE STATE AUTOMATA AND UNILINEAR GRAMMARS

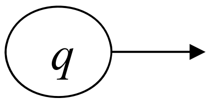
they define exactly the same languages

First we show how to derive an equivalent grammar from an automaton

The grammar: nonterminal symbols are the states $\boldsymbol{Q}$ of the automaton
axiom: the initial state



$\Rightarrow$ $\boxed{q \rightarrow ar}$



(q final state) $\Rightarrow$ $\boxed{\forall \; q \in F, \;\; q \rightarrow \varepsilon}$

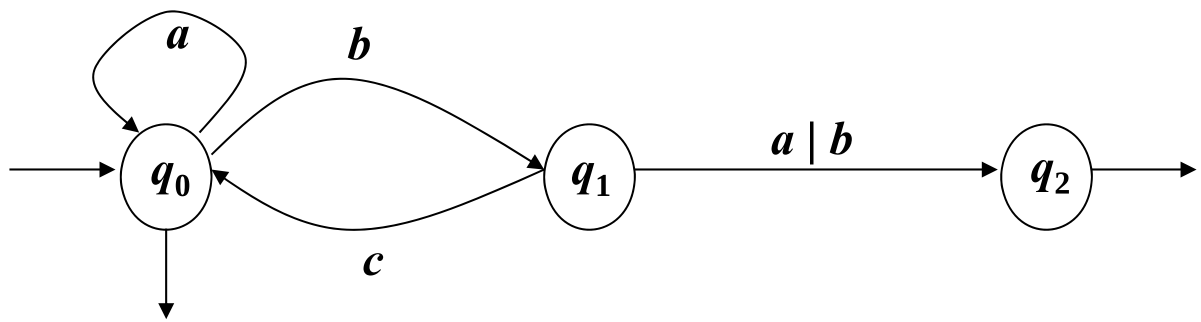
If a non-nullable grammar is preferred:

$\boxed{q \rightarrow ar \mid a}$

there is a one-to-one map: computations of the automaton $\leftrightarrow$ derivations of the grammar

string x is accepted by the automaton iff $\exists$ a derivation $q_0 \stackrel{*}{\Rightarrow} x$

Example



$$q_0 \to aq_0 \mid bq_1 \mid \varepsilon$$
$$q_1 \to cq_0 \mid aq_2 \mid bq_2$$
$$q_2 \to \varepsilon$$

derivation of string *bca*

$$q_0 \Rightarrow bq_1 \Rightarrow bcq_0 \Rightarrow bcaq_0 \Rightarrow bca\varepsilon = bca$$

REVERSE TRANSFORMATION: GRAMMAR ⇒ AUTOMATON

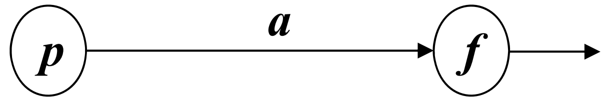
	right-linear grammar	*finite automaton*
1.	Nonterm. alphabet ***V***	set of states ***Q*** = ***V***
2.	axiom S	initial state q_0 = S
3.	$\boxed{p \to aq,\ a \in \Sigma \text{ and } p, q \in V}$	$p \xrightarrow{a} q$
4.	$\boxed{p \to q \text{ where } p, q \in V}$	$p \xrightarrow{\varepsilon} q$
5.	$\boxed{p \to \varepsilon}$	final state $p \rightarrow$

notice that in general the automaton will be nondeterministic

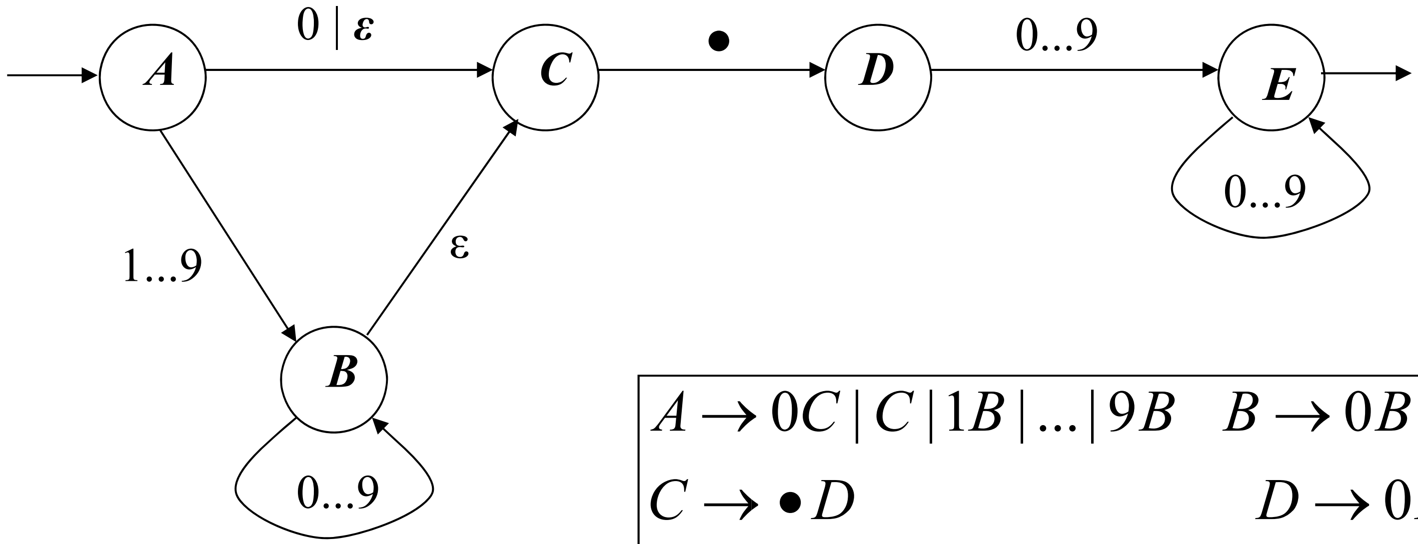
DERIVATION OF THE GRAMMAR ⇔ COMPUTATION OF THE AUTOMATON
therefore the two models define the same language

⇒ A LANGUAGE IS ACCEPTED BY A FINITE AUTOMATON IF AND ONLY IF IT IS GENERATED BY A UNILINEAR GRAMMAR

If the grammar includes terminal rules of type $\boldsymbol{p \to a}$, $\boldsymbol{a \in \Sigma}$,
The automaton will contain an additional state $\boldsymbol{f}$, with $\boldsymbol{f}$ final, and the transition:



Example – Equivalence right-linear grammars – finite automata



$$A \to 0C \mid C \mid 1B \mid \ldots \mid 9B \quad B \to 0B \mid \ldots \mid 9B \mid C$$

$$C \to \bullet D \qquad D \to 0E \mid \ldots \mid 9E$$

$$E \to 0E \mid \ldots \mid 9E \mid \varepsilon \qquad \text{axiom } A$$

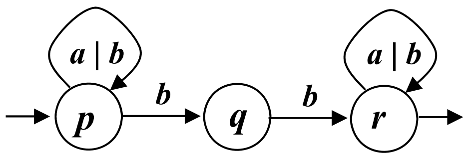
DEFINITION: an automaton is ***ambiguous*** if it accepts a sentence with distinct computations

one-to-one match between computations (automata) and derivations (grammars) $\Rightarrow$
an automaton is ambiguous $\Leftrightarrow$ corresponding right-linear grammar is ambiguous

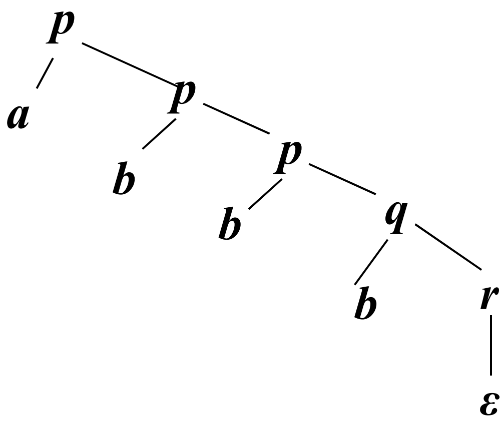
NB: for linear grammar there is a 1-to-1 match between derivations and syntax trees

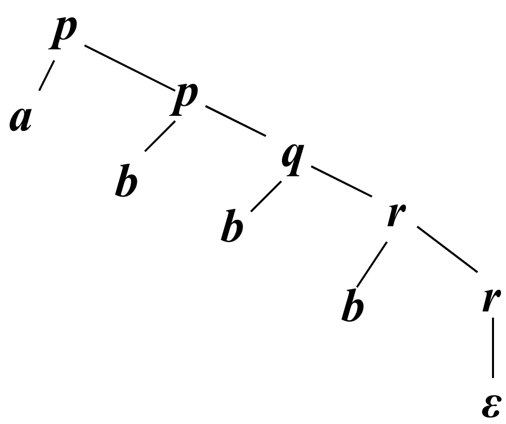
Example (continued) – Search of string *bb* in the text *abbb*

$$\boldsymbol{p \rightarrow ap \mid bp \mid bq}$$
$$\boldsymbol{q \rightarrow br}$$
$$\boldsymbol{r \rightarrow ar \mid br \mid \varepsilon}$$



(a | b)* bb (a | b)*





GRAMMAR ⇒ AUTOMATA IN CASE OF LEFT-LINEAR GRAMMARS

a variation of the case discussed above – see textbook §3.5.6

left-lin.gr. G ⇒(reverse)⇒ G_R(right-lin) ⇒ automaton for L_R ⇒(reverse)⇒ automaton for $L(G)$

FROM AUTOMATA TO REGULAR EXPRESSIONS DIRECTLY
THE BMC (Brzozowski & McCluskey) METHOD

Assumptions

- unique initial state i without incoming arcs, and
- unique final state t without outgoing arcs

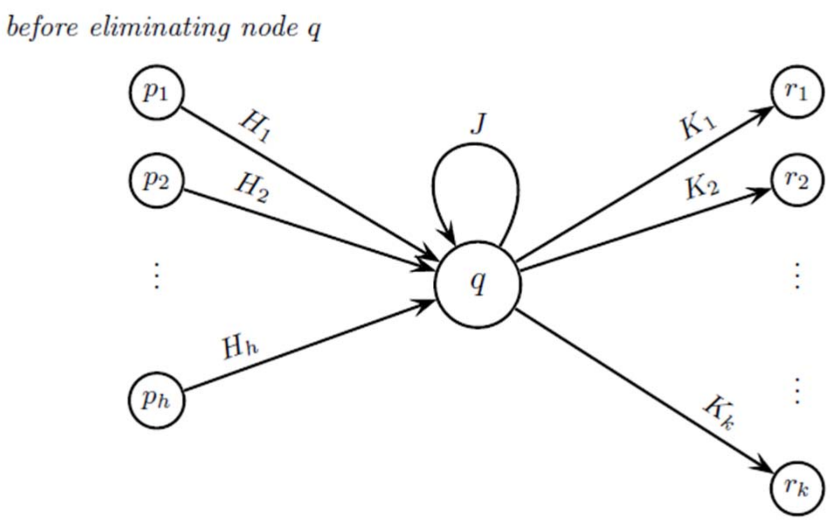
(otherwise: add initial and final states connected through suitable spontaneous moves)

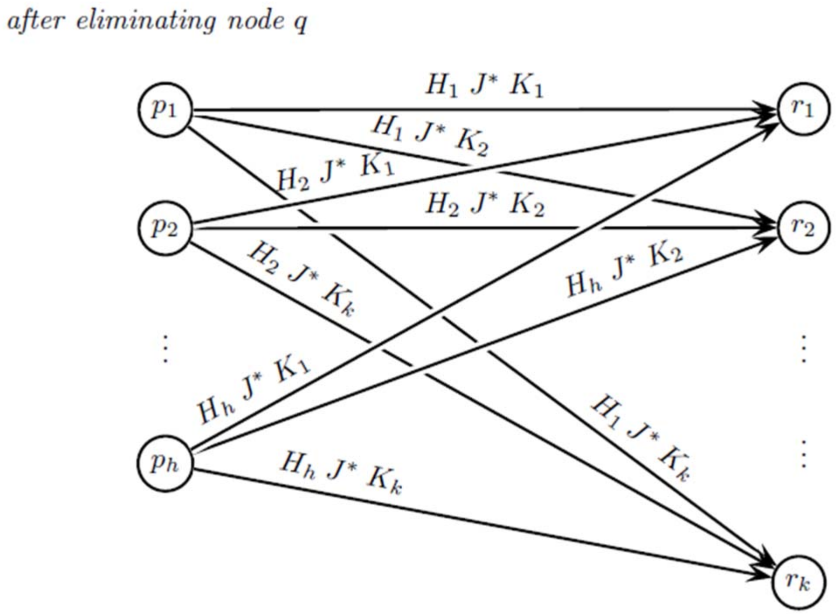
states different from i and t are called ***internal***

a new **generalized automaton** is built :
equivalent to the initial one, but with arcs labeled by regular expressions

internal states are progressively eliminated
compensating moves are added, labeled by r.e. that preserve the accepted language
until only states i and t are left

then the r.e. *e* labeling the transition $i \xrightarrow{e} t$ is the e.r. of the language



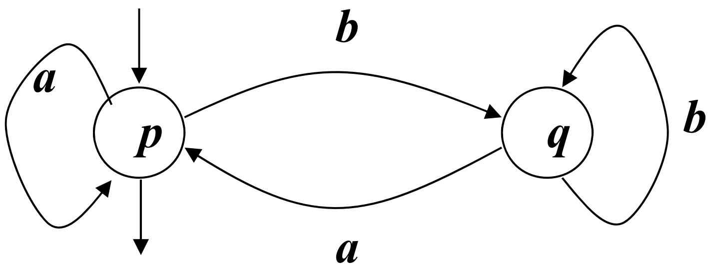


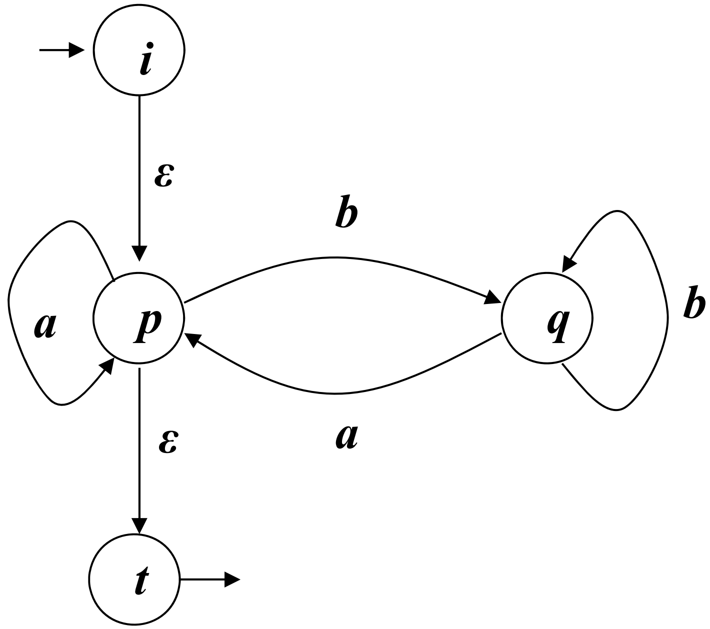
for each pair of states $\boldsymbol{p_i}\, \boldsymbol{r_j}$ a compensating transition: $p_i \xrightarrow{H_i J^* K_j} r_j$ (some p_i, r_j may coincide)

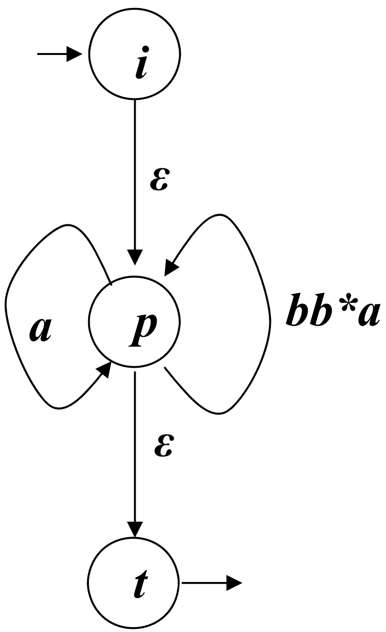
any resulting «parallel» transitions $p \overset{e_1}{\rightarrow} r$ and $p \overset{e_2}{\rightarrow} r$ «merged» into $p \xrightarrow{e_1 \mid e_2} r$

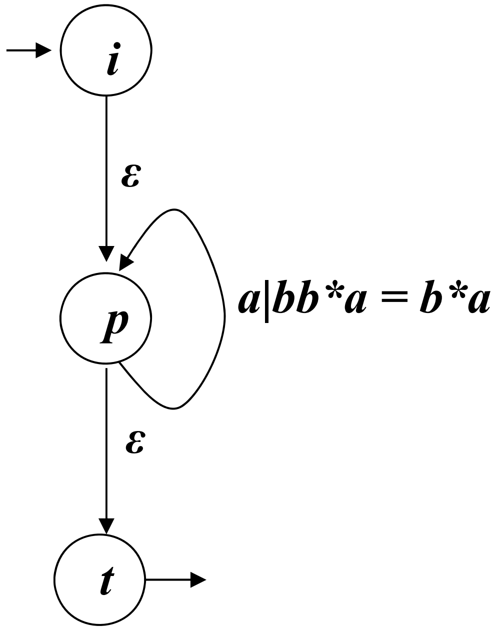
changing the order in the internal state elimination moves
leads to obtain formally distinct, but equivalent regular expressions
pay attention when solving exercises: simplify r.e.'s whenever possible

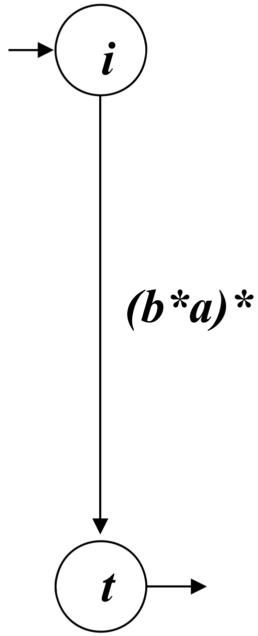
Example – application of BMC











ELIMINATION OF NONDETERMINISM

for efficiency, the final, implemented version of a finite recognizer must be deterministic

PROPERTY:
every nondeterministic automaton can be transformed into a deterministic one, and
(corollay) every unilinear grammar admits an equivalent nonambiguous grammar

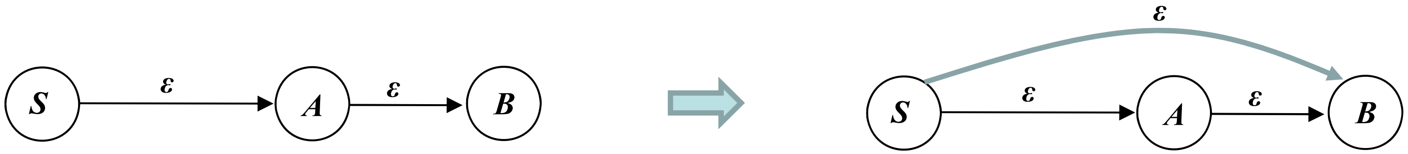
determinization of a finite automaton conceptually separated in two parts
(both steps can be replaced by the Berry-Sethi construction, to be presented later)

1. elimination of spontaneous moves: move sequences that include spontaneous moves are replaced by ***scanning moves*** (non-ε arcs)

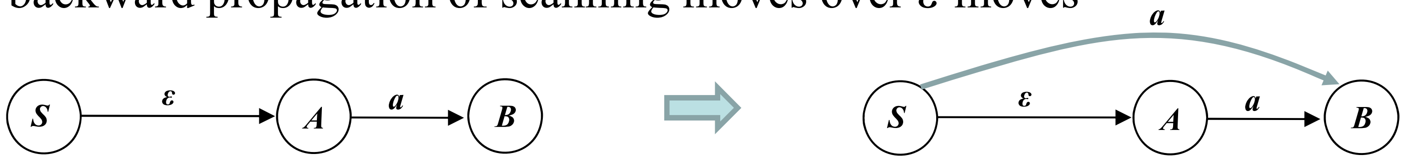
2. replacement of several nondeterministic (scanning) transitions by a single one (accessible subset construction – the new states are subsets of the initial state set): we do not cover that; see textbook §3.7.1 (also covered by previous courses)

First phase: elimination of ε-moves – a 4-steps procedure

1: transitive closure of ε-moves



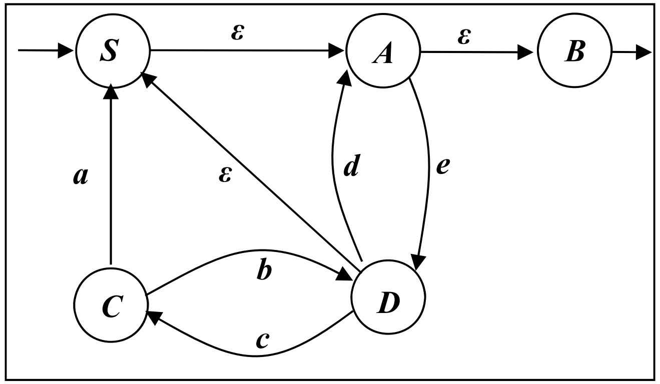
2: backward propagation of scanning moves over ε-moves



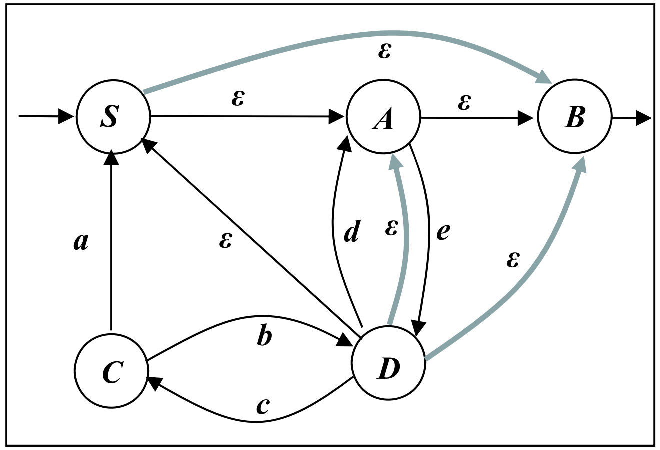
3: new final states: backward propagation of the «finality condition» for final states reached by ε-moves (antecedent states become also final)

4: clean-up of ε-moves and of useless states

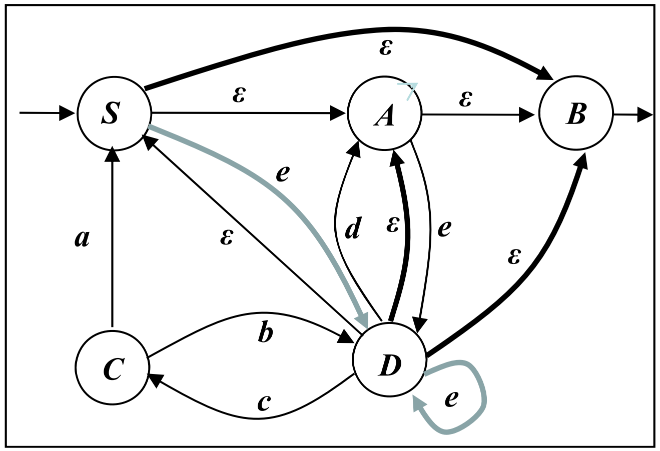
Example



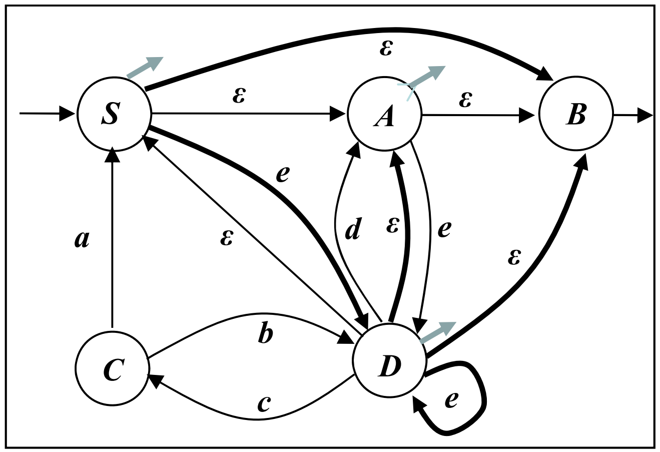
1



2



3



4