From Regular Expressions to Recognizing automata

Prof. A. Morzenti

From Regular Expressions to Recognizing automata

Various algorithms, they differ in the kind of automaton and in the size of the result

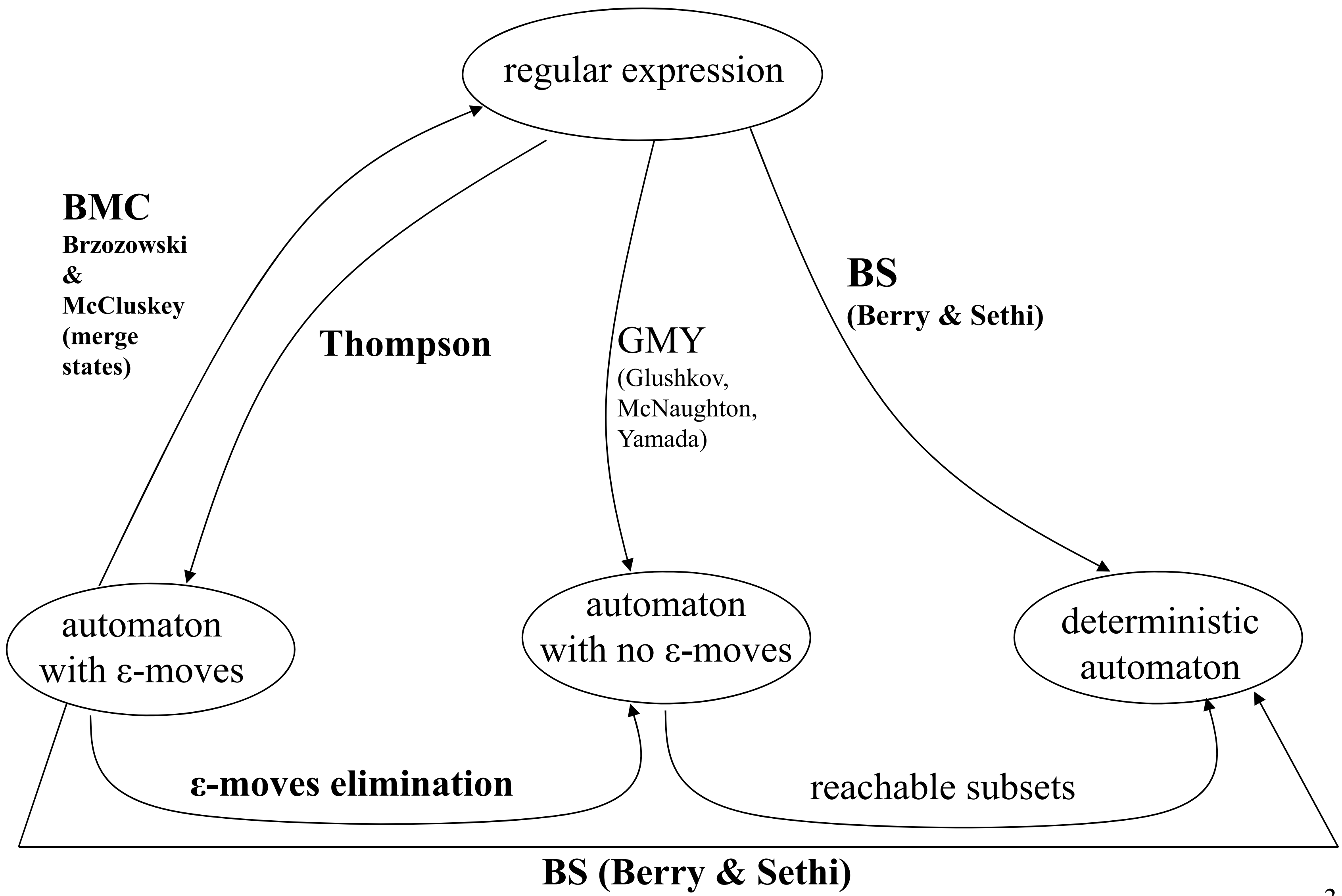
the textbook reports three methods (we will discuss (1) and (3)):

1) THOMPSON (or structural) method
   - builds the recognizers of subexpressions
   - combines them through spontaneous moves
   - resulting automata have (several) **ε-moves** and are in general **nondeterministic**

2) GLUSHKOV, MC NAUGHTON and YAMADA (GMY) method
   - builds a **nondeterministic automaton having no spontaneous moves**
   - size is less than Thompson's

3) BERRY & SETHI (BS) method
   - builds a **deterministic** automaton
   - not necessarily minimal

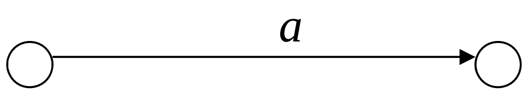
(1) and (2) can be combined with determinization algorithms,
(3) with minimization algorithms


regular expression
BMC
Brzozowski
&
McCluskey
(merge
states)
Thompson
GMY
(Glushkov,
McNaughton,
Yamada)
BS
(Berry & Sethi)
automaton
with ε-moves
automaton
with no ε-moves
deterministic
automaton
ε-moves elimination
reachable subsets
BS (Berry & Sethi)

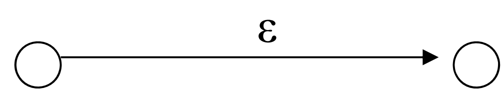
THOMPSON's STRUCTURAL METHOD

1) based on a systematic mapping between r.e. and recognizing automata
2) every portion of automaton must have a unique initial and final state

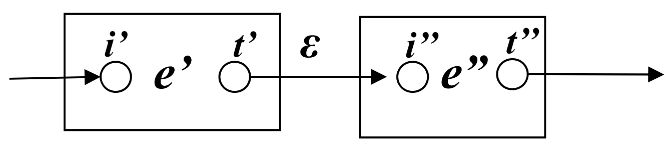
r.e. of type : a with $a \in \Sigma$



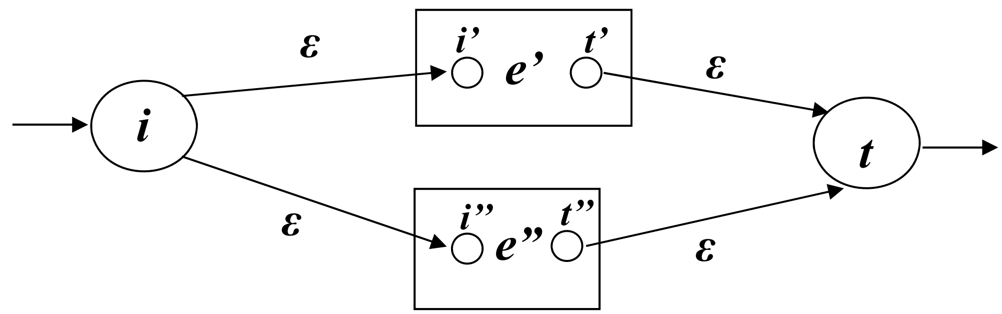
r.e. of type: ε



concatenation ***e' · e''*** of two r.e. *e'* and *e''*

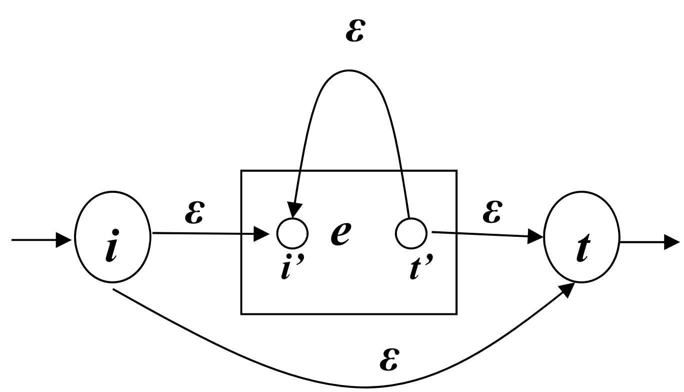


Union of two r.e. *e*’ and *e*”



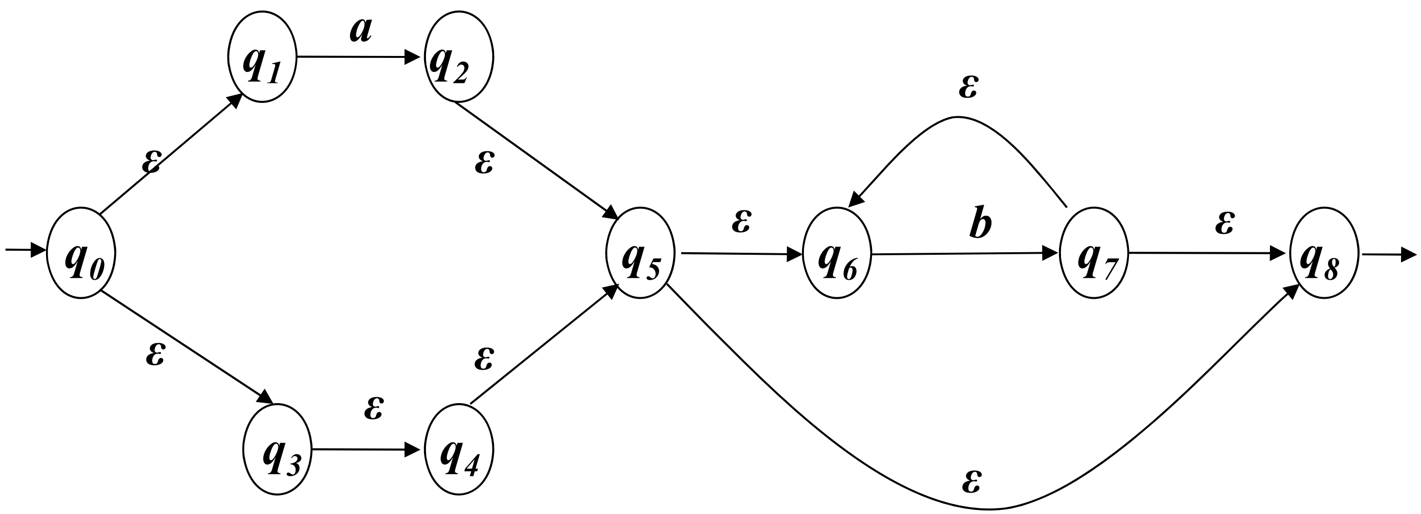
Star ***e**** of a r.e. ***e***

ε



Example

$$\boxed{(a \cup \varepsilon).b^*}$$



Before discussing the Berry – Sethi method, we introduce the ...

... LOCALLY TESTABLE languages, also called LOCAL (***LOC***)

LOC is a ***proper sub***family of regular languages ($\boldsymbol{LOC \subset REG}$, $\boldsymbol{LOC \neq REG}$)

DEFINITIONS: given a language L of alphabet Σ, define (assuming $\boldsymbol{a, b} \in \Sigma$ and $\boldsymbol{x, y} \in \Sigma^*$)

set of Initial chars $\quad Ini(L) = \{a \mid ax \in L\}$

set of Finishing chars $\quad Fin(L) = \{b \mid xb \in L\}$

set of Digrams $\quad Dig(L) = \{ab \mid xaby \in L\}$

complement of Digrams $\quad \overline{Dig}(L) = \Sigma^2 \setminus Dig(L)$

Example: $L_1 = (abc)^*$ has local sets $Ini(L_1)=\{a\}$, $Fin(L_1)=\{c\}$, $Dig(L_1)=\{ab, bc, ca\}$

Definitions of *Ini*, *Dig*, *Fin*, are provided similarly for individual strings

Ex.: for string $x=abc$ we have $Ini(x)=\{a\}$, $Dig(x)=\{ab, bc\}$, $Fin(x)=\{c\}$

$L_1 = (abc)^*$ includes all strings obtainable from its *Ini*, *Dig* e *Fin* (NB they are $\neq \varepsilon$)

Ex.: string *abcabc* obtained by composing, like in a «domino game», *a*, *ab*, *bc*, *ca*, *ab*, *bc*, *c*

NB: for every language L (even for non-regular languages) it holds

$$L \setminus \{\varepsilon\} \subseteq \{\ x \ |\ Ini(x) \in Ini(L) \ \wedge\ Dig(x) \subseteq \mathrm{Dig}(L) \ \wedge\ Fin(x) \in Fin(L)\ \}$$

Because, trivially, every sentence of L

- starts (resp. ends) with a char $c \in Ini(L)$ (resp. $c \in Fin(L)$), and
- its digrams are included in those of the language

A language $L \in LOC$ includes ***all*** the strings generated by the three local sets,
i.e., the language contains ***all and only*** the strings that can be built from *Ini*, *Fin*, and *Dig*
(plus, possibly, ε)

$$L \in LOC \text{ iff } L \setminus \{\varepsilon\} = \{\, x \mid Ini(x) \in Ini(L) \wedge Dig(x) \subseteq \mathrm{Dig}(L) \wedge Fin(x) \in Fin(L) \}$$

Instead, $L \notin LOC$ if it does not include all strings generated from *Ini*, *Dig*, and *Fin*
i.e. $\exists$ a string $x \notin L$ that is generated from the local sets of L

NB: the definition provides a ***necessary condition*** for a language to be local
and therefore a method for proving that a language ***is NOT*** local
(to prove that language L ***is not*** local, exhibit a *witness*:
a string $x \notin L$ s.t. $Ini(x) \in Ini(L)$, $Fin(x) \in Fin(L)$ and $Dig(x) \subseteq Dig(L)$)

Ex.: $L_1=(abc)^*$ is local: $Ini(L_1)=\{a\}$, $Fin(L_1)=\{c\}$, $Dig(L_1)=\{ab, bc, ca\}$

All strings obtained from *Ini*, *Fin*, *Dig* are included in L_1

Example of **non**local regular language

L_2 is ***strictly included*** in the set of strings generated from its local sets *Ini*, *Dig*, *Fin*

Indeed L_2 does not include strings of odd length nor strings with a *b* surrounded by *a*'s

$$L_2 = b(aa)^+ b$$
$$Ini(L_2) = Fin(L_2) = \{b\}$$
$$Dig(L_2) = \{aa, ab, ba\}$$
$$\overline{Dig}(L_2) = \{bb\}$$
$$baab, baaaab \in L_2$$
$$baaab \notin L_2 \quad baabab \notin L_2 \quad \ldots$$

L_1 and L_2 are regular, L_1 is local, L_2 is not local

Therefore $\boldsymbol{LOC} \subset \boldsymbol{REG}$, $\boldsymbol{LOC} \neq \boldsymbol{REG}$, inclusion is ***strict***

For every non-local regular language *L* there exists a superset of it, L_{LOC}, which is local:

It contains **all** strings obtainable through *Ini*(*L*), *Dig*(*L*), *Fin*(*L*) (a sort of trans. closure …)

NB: the property of locality is determined by (presence or absence of) non-empty strings:

$\Rightarrow$ presence of ε in the language is immaterial

Ex.: $(abc)^*$ and $(abc)^+$ are **both** local

IT IS VERY SIMPLE TO BUILD A RECOGNIZER OF A LOCAL LANGUAGE

it scans the string, and checks that:

* the first char $\in$ *Ini*
* every pair of consecutive chars $\in$ *Dig*
* the last char $\in$ *Fin*

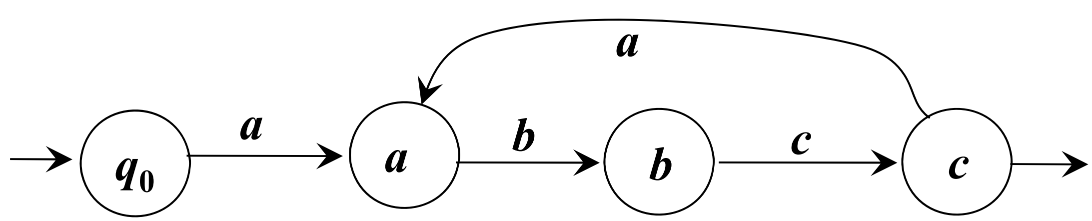
string analysis from left to right using a *shifting window* of length two, implemented by a very simple deterministic automaton: what must be «remembered»?

only the last read char ($\Rightarrow$ a one-to-one mapping between $\boldsymbol{Q}$ and Σ)

construction of the recognizer of language $L \in LOC$ starting from *Ini*, *Fin*, *Dig*

1. a unique initial state $\boldsymbol{q_0}$
2. set of states $\boldsymbol{Q} = \Sigma \cup \{\boldsymbol{q_0}\}$ (all states but the initial one are labeled by an element of Σ)
3. final states $\boldsymbol{F} = \boldsymbol{Fin}$; if $\varepsilon \in L$ then $\boldsymbol{F}$ also includes $\boldsymbol{q_0}$
4. transition function $\boldsymbol{\delta}$: $\forall \boldsymbol{a} \in \mathbf{Ini}\ \boldsymbol{\delta(q_0, a) = a}$; $\quad \forall \boldsymbol{xy} \in \boldsymbol{Dig}\ \ \boldsymbol{\delta(x, y) = y}$

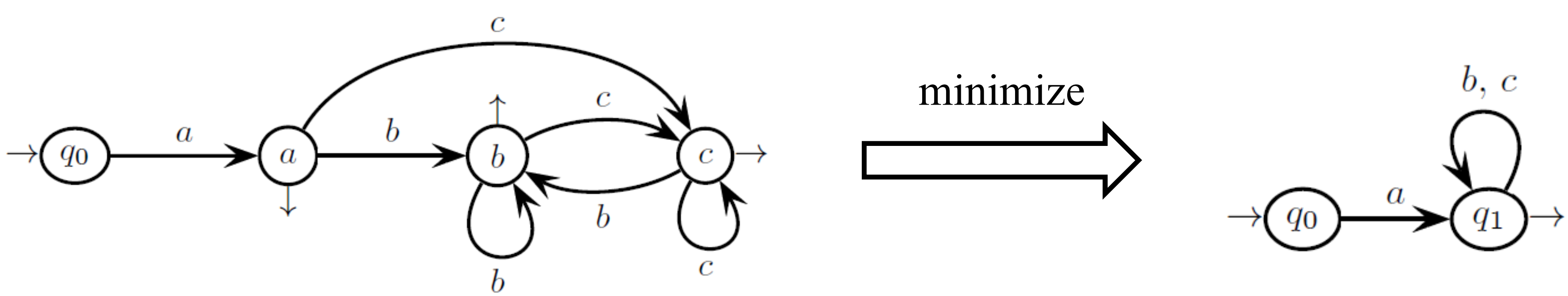
Example: $\boldsymbol{L_1 = (abc)^+}$ $\quad \boldsymbol{Ini(L_1) = \{a\}}$ $\quad \boldsymbol{Fin(L_1) = \{c\}}$ $\quad \boldsymbol{Dig(L_1) = \{ab, bc, ca\}}$



(from previous construction)
SOME CRITERIA FOR PROVING THAT A LANGUAGE *L* ***IS*** LOCAL

1. *L* is accepted by an automaton satisfying conditions 1 – 4 in previous slide (it is called ***normalized*** local automaton)

however such an automaton might be NOT minimal



hence a second, less restrictive, sufficient condition:

2. *L* is accepted by a (possibly minimal) automaton obtained from the normalized local automaton by merging indistinguishable states

BERRY & SETHI DETERMINISTIC RECOGNIZER

(we only illustrate it: see textbook, §3.8.2.4 for a complete explanation)

Let e the starting r.e. (of alphabet Σ): e.g. $e = (a \mid bb)^* (ac)^+$

e' its ***numbered version*** (of alphabet Σ_N): e.g. $e' = (a_1 \mid b_2 b_3)^* (a_4 c_5)^+$

We consider expression $e' \dashv$ which includes the end-of-text mark $\dashv$

We define, for each symbol a of e', the set of ***Followers*** of a, $Fol(a)$

It is the set of symbols that, in the strings $\in L(e' \dashv)$, can follow a

Essentially, the same information as $Dig(e' \dashv)$

$$Fol(a) = \{ b \mid ab \in Dig(e' \dashv) \}$$

Hence $\dashv \in Fol(a)$ for every $a \in Fin(e')$

Ex.: for $e' = (a_1 \mid b_2 b_3)^* (a_4 c_5)^+ \dashv$ we have

$Fol(a_1)=\{a_1, b_2, a_4\}$ $\quad$ $Fol(b_2)=\{b_3\}$ $\quad$ $Fol(b_3)=\{a_1, b_2, a_4\}$

$Fol(a_4)=\{c_5\}$ $\quad$ $Fol(c_5)=\{a_4, \dashv\}$

Construction of the deterministic recognizer of Berry-Sethi

Every state is (corresponds to) a subset of $\Sigma_N \cup \{\dashv\}$

It contains the symbols that one can ***expect as next input***

Therefore final states are those that include (possibly among others) the end-mark $\dashv$

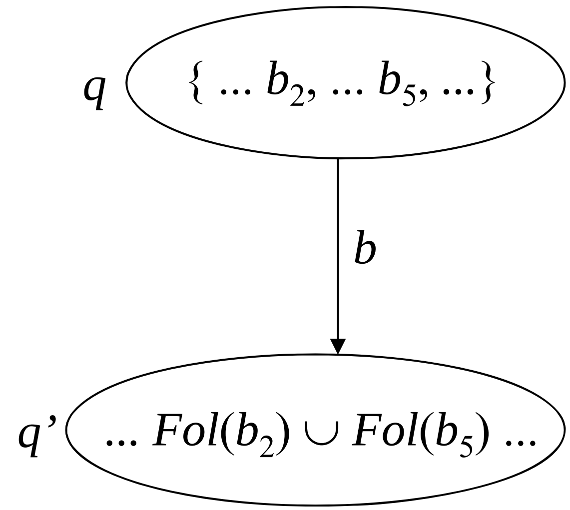
The ***initial state*** is the set $\boldsymbol{Ini(e' \dashv)}$

States are generated from the initial one, adding transitions and new states

Until a fixed point is reached (no new state can be generated)

During construction the transition function δ is viewed as a set of transitions $q \overset{a}{\rightarrow} q'$

BS ALGORITHM



$q_0 := Ini(e' \dashv)$;   mark q_0 as *not visited*
$Q := \{q_0\}$
$\delta := \varnothing$
while there exists in Q a non-visited state q **do**
    mark q as visited
    for each symbol $b \in \Sigma$ **do**
        $\boxed{q' := \bigcup_{\forall\, b_i \in q} Fol(b_i)}$
        if $q' \neq \varnothing$ **then**
            if $q' \notin Q$ **then**
                mark q' as *not visited*
                $Q := Q \cup \{\ q'\ \}$
            end if
            $\delta := \delta \cup \{\ q \xrightarrow{b} q'\ \}$
        end if
    end do
end do

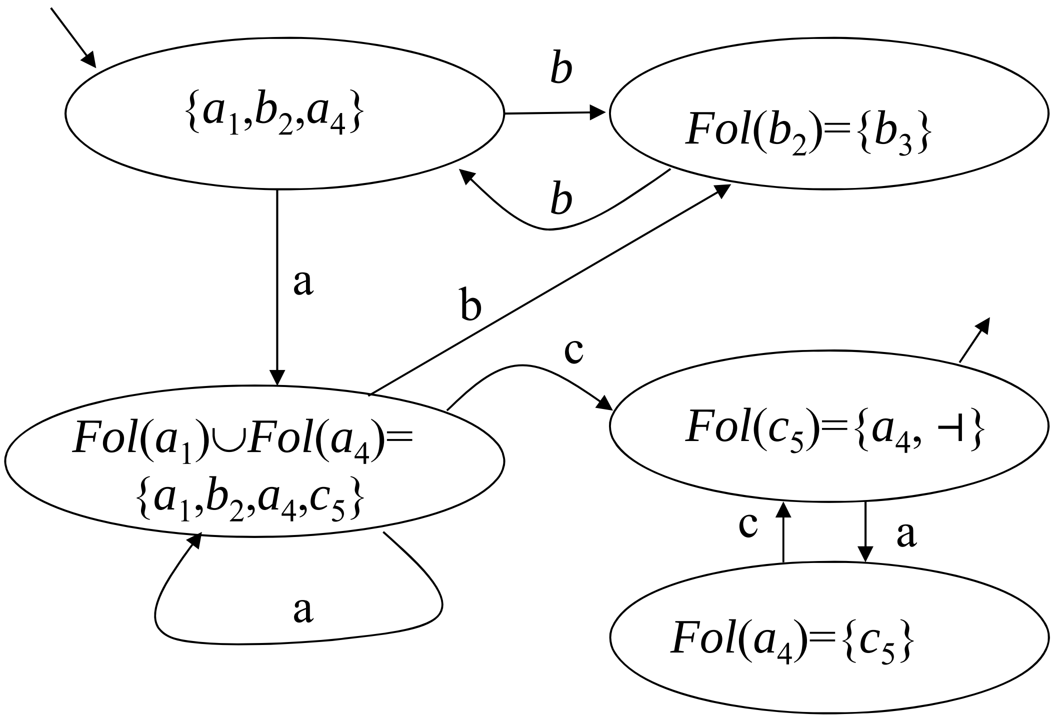
Example

$e = (\ a \mid bb\ )^*\ \ (ac)^+$

$e' \dashv = (\ a_1 \mid b_2 b_3\ )^*\ \ (a_4 c_5)^+ \dashv$

$Ini(e' \dashv\ ) = \{\ a_1, b_2, a_4\ \}$

x	$Fol(x)$
a_1	a_1, b_2, a_4
b_2	b_3
b_3	a_1, b_2, a_4
a_4	c_5
c_5	$a_4, \dashv$



the resulting automaton is deterministic
but it can be ***non*-minimal**
(because of the numbering of symbols)

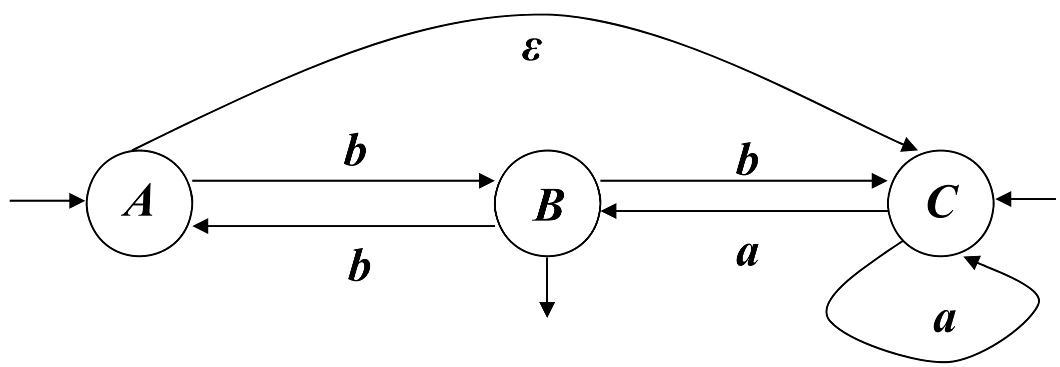
USING THE BS ALGORITHM FOR AUTOMATA DETERMINIZATION

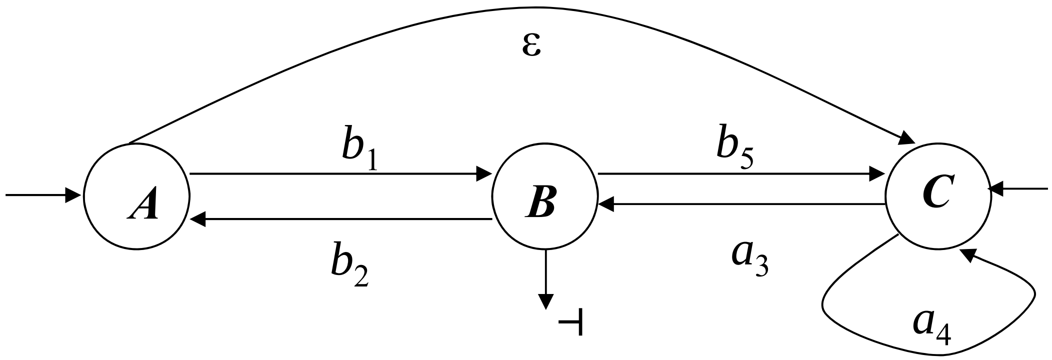
BS algorithm used for determinizing a nondeterministic automaton *N* with ε-arcs

1. number the non-ε arcs of ***N***, obtaining a numbered version ***N'*** ; add an endmark '⊣' on darts exiting the final states
2. compute for ***N'*** the local sets *Ini* and *Fol* (using rules similar to those for a r.e.)
3. apply the BS construction, thus obtaining an automaton ***M***

the resulting automaton is deterministic, though possibly non-minimal

Example

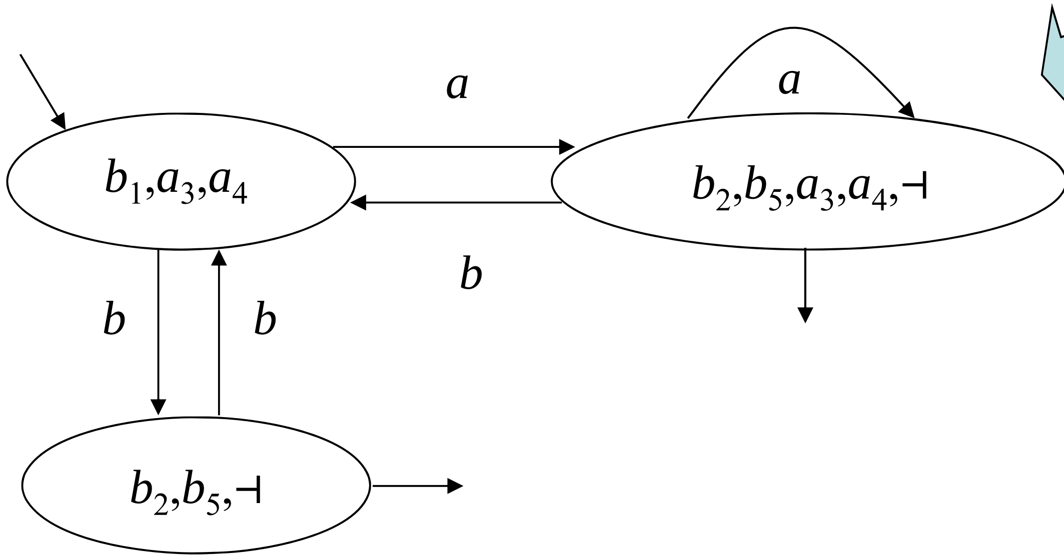




$Ini(L(N')\text{-|}) = \{b_1, a_3, a_4\}$

$\varepsilon a_3 = a_3, \varepsilon a_4 = a_4$

x	$Fol(x)$
b_1	$b_2, b_5, \text{-|}$
b_2	b_1, a_3, a_4
a_3	$b_2, b_5, \text{-|}$
a_4	a_3, a_4
b_5	a_3, a_4



REGULAR EXPRESSIONS WITH (1) COMPLEMENT AND (2) INTERSECTION

both topics covered by previous courses so we go through them quickly

(1) CLOSURE OF REG UNDER COMPLEMENT AND INTERSECTION

$$\boxed{\text{If } L, L', L'' \in REG \text{ then } \neg L \in REG \quad L' \bigcap L'' \in REG}$$

Proved by constructing the recognizer for the complement language $\neg \boldsymbol{L} = \boldsymbol{\Sigma^* \setminus L}$

starting from a ***deterministic*** recognizer M for L

ALGORITHM: construction of the deterministic recognizer $\boldsymbol{M'}$ for the complement

We extend $\boldsymbol{M=<Q, \Sigma, \delta, q_0, F>}$ with the ***error*** or ***sink*** state $p \notin \boldsymbol{Q}$ and the arcs to and from it

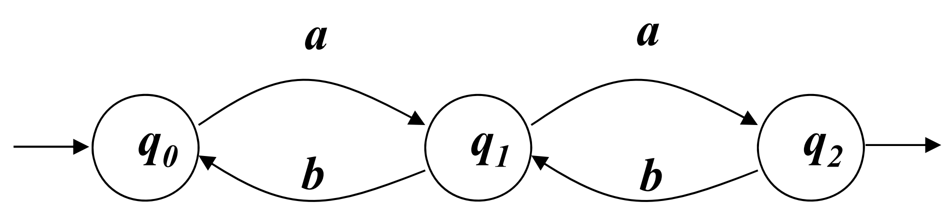
1. $Q' = Q \cup \{p\}$

2. $\delta'(q, a) = \delta(q, a)$ if $\delta(q, a)$ is defined, otherwise $\delta'(q, a)=p$; $\delta'(p, a)=p \ \ \forall a \in \Sigma$

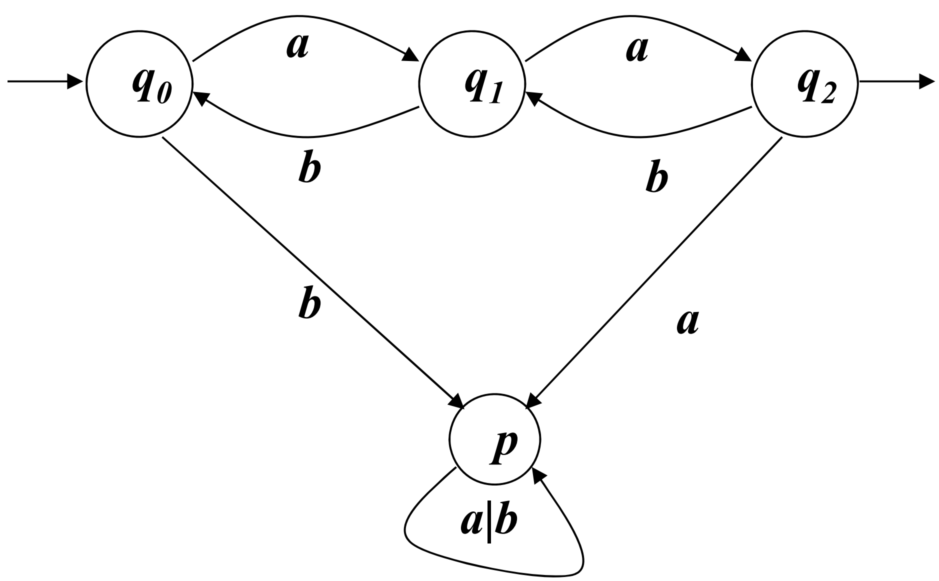
3. Switch initial and final states: $\boldsymbol{F' = (Q \setminus F) \cup \{ p \}}$

Example: automaton for the complement language

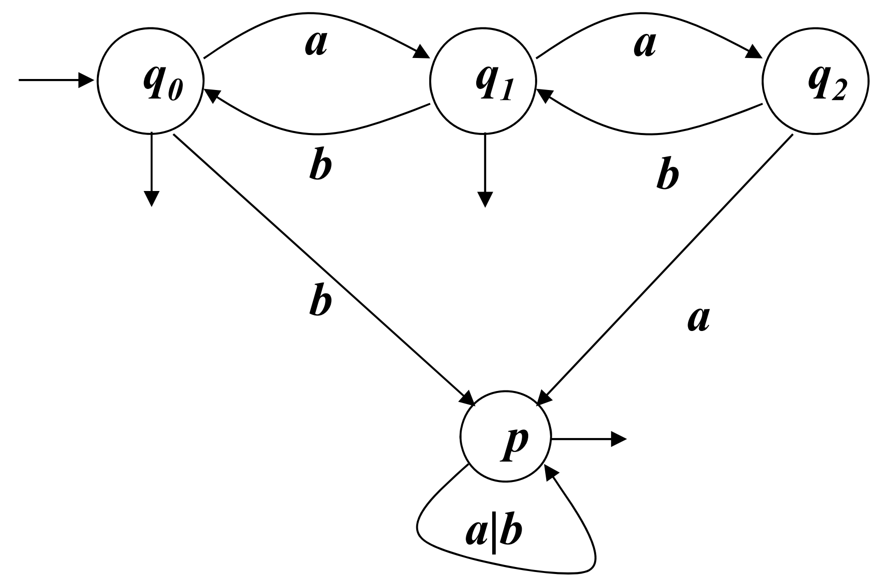
original automaton:



automaton with the sink state added:



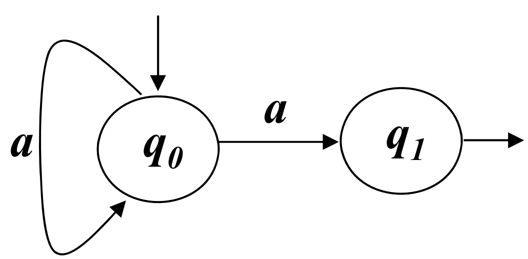
complement automaton
(final states switched):



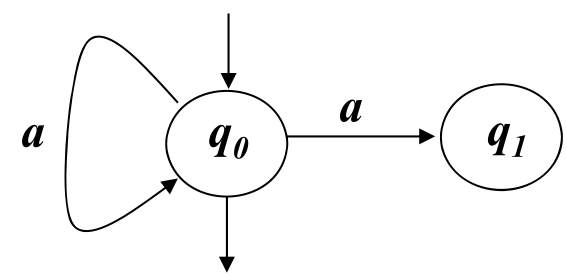
NB: the starting automaton ***M*** **must** be deterministic
otherwise the language accepted by the complement automaton ***M'*** migh not be disjointed
and the obvious property $L \cap \neg L = \varnothing$ would be violated

A nondeterministic automaton can have, for a string $x \in L$,
one accepting computation and a non-accepting one
in the complement automaton ***M'*** this non-accepting computation of ***M*** would be accepting

Example:



original automaton ***M*** (***nondeterministic***)



(pseudo) complement automaton ***M'***

The pseudo complement automaton accepts string ***a***, that also belongs to the original language

2) RECOGNIZER FOR THE INTERSECTION OF TWO REGULAR LANGUAGES

one could use the property of closure of REG under complement and union
to exploit the De Morgan identity $L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$ and therefore:

- build the deterministic recognizers of L_1 and L_2
- derive those of the complement languages $\neg L_1$ and $\neg L_2$
- build the recognizer for the union (using the Thomson method)
- make the automaton deterministic
- derive the complement automaton

There is a more direct method

(CARTESIAN) PRODUCT AUTOMATON

Quite a common method:
Allows one to simulate the simultaneous execution of two automata

We assume the two automata without ε-moves but not necessarily deterministic

The state set of the product automaton ***M*** is the cartesian product of the state sets of *M'* and *M''*

A state is a pair ***<q', q''>***, with $q' \in Q'$ and $q'' \in Q''$

definition of transition function:

$$\boxed{<q', q''> \xrightarrow{a} <r', r''> \quad \text{if and only if} \quad q' \xrightarrow{a} r' \wedge q'' \xrightarrow{a} r''}$$

Initial states *I* of *M* are also the cartesian product $I = I' \times I''$

Final states are also the product $F = F' \times F''$

NOTE: The method can be applied to other set-theoretical operations (e.g. for union: final states of ***M*** are those state pairs where at least one of the two states is final)

Example – Intersection and product machine for the two languages of strings containing, respectively, substring ***ab*** and ***ba***

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

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