Dall’espressione regolare all’automa riconoscitore

Prof. A. Morzenti

aa 2008-2009

DALL'ESPRESSIONE REGOLARE ALL'AUTOMA RICONOSCITORE

Diversi algoritmi, si differenziano per proprietà dell'automa prodotto (deterministico o indeterministico, con o senza mosse spontanee, dimensioni), e per complessità algoritmica della costruzione.

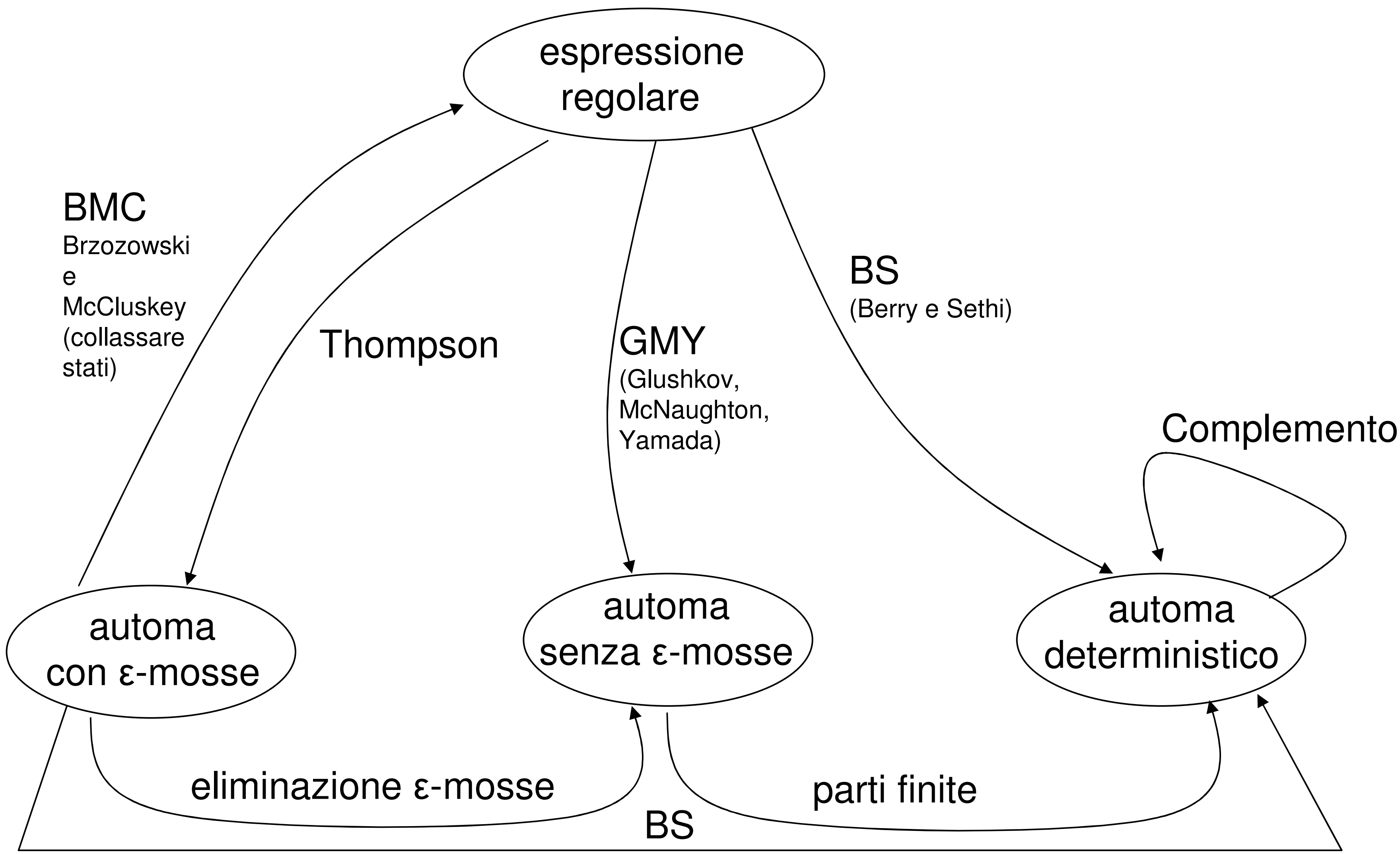
VEDIAMO TRE METODI:

1) METODO DI THOMSON (o strutturale): - decompone la e.r. nelle sue sottoespressioni (fino ad arrivare agli elementi atomici dell'alfabeto),
   - costruisce i riconoscitori delle varie sottoespressioni
   - li connette in reti di riconoscitori che realizzano gli operatori di unione, concatenamento, stella e croce
   - gli automi hanno (molte) ε-mosse e sono in generale nondeterministici

2) METODO DI GLUSHKOV, MC NAUGHTON e YAMADA (GMY): costruisce un automa nondeterministico senza mosse spontanee, di dimensioni maggiori di quello del metodo precedente

3) METODO DI BERRY E SETHI (BS): costruisce un automa deterministico

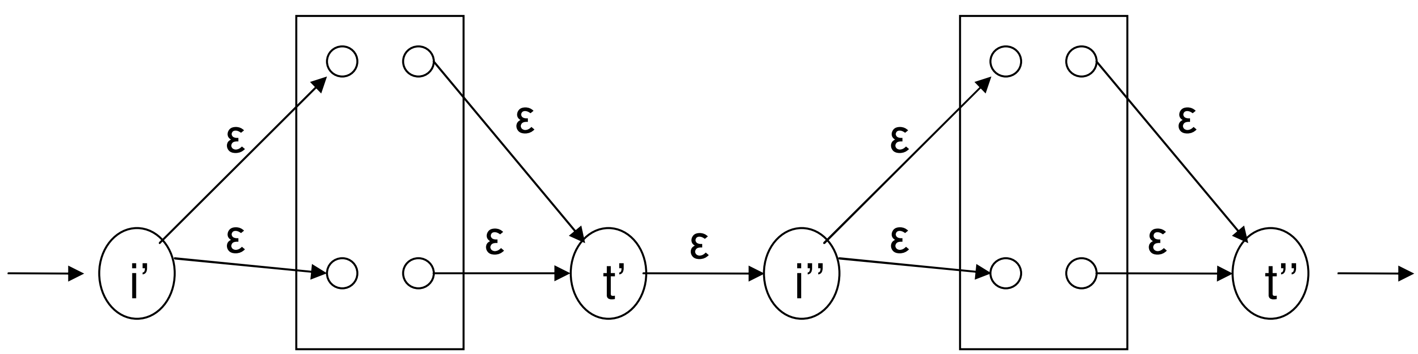
I PRIMI DUE METODI sono COMBINABILI con algoritmi di determinizzazione.


espressione regolare
BMC
Brzozowski e McCluskey (collassare stati)
Thompson
GMY
(Glushkov, McNaughton, Yamada)
BS
(Berry e Sethi)
Complemento
automa con ε-mosse
automa senza ε-mosse
automa deterministico
eliminazione ε-mosse
parti finite
BS

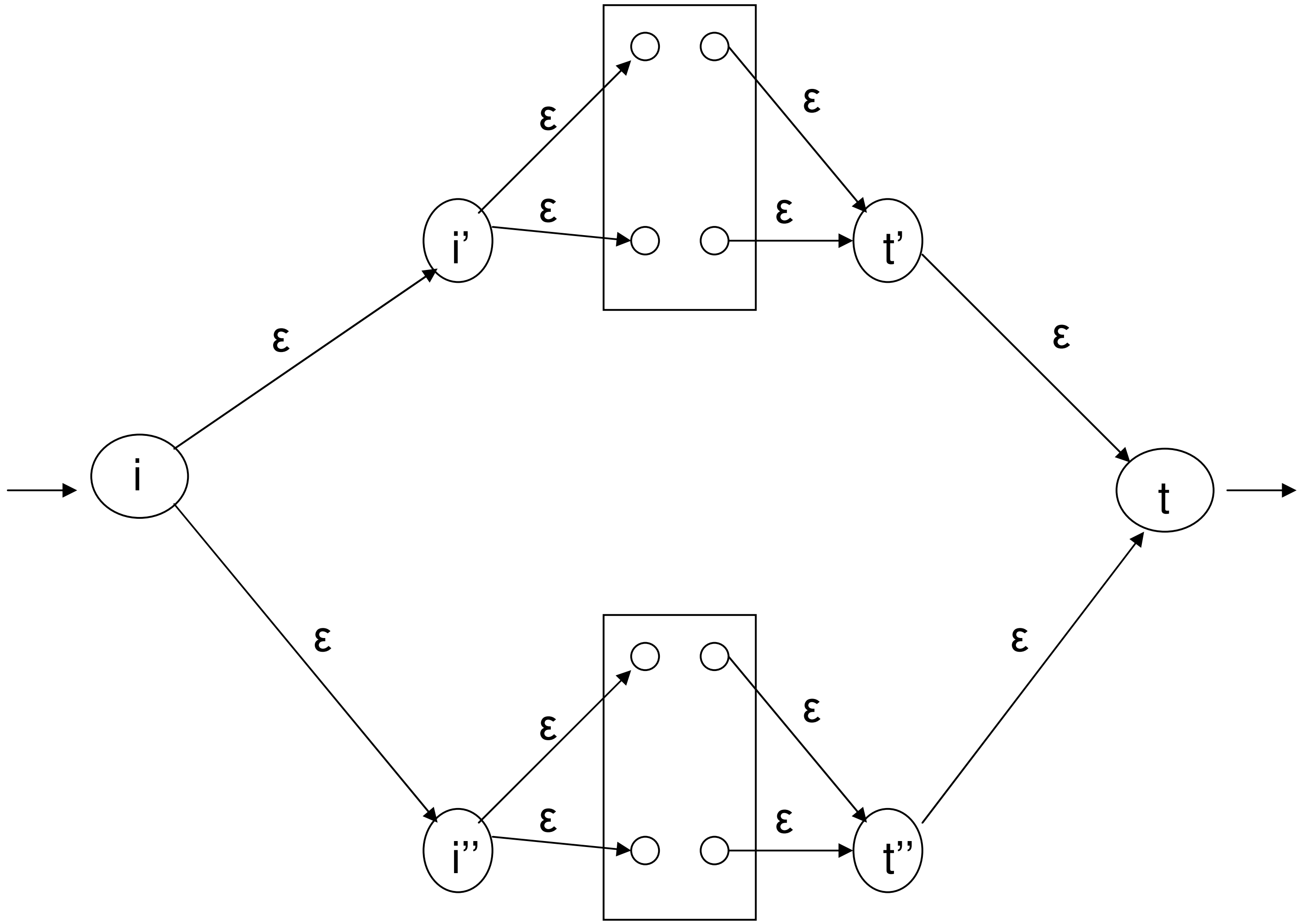
METODO STRUTTURALE O DI THOMPSON

1) sfrutta le regole di corrispondenza tra e.r. e automi riconoscitori
2) ogni pezzo di automa deve avere un solo stato iniziale e uno solo finale

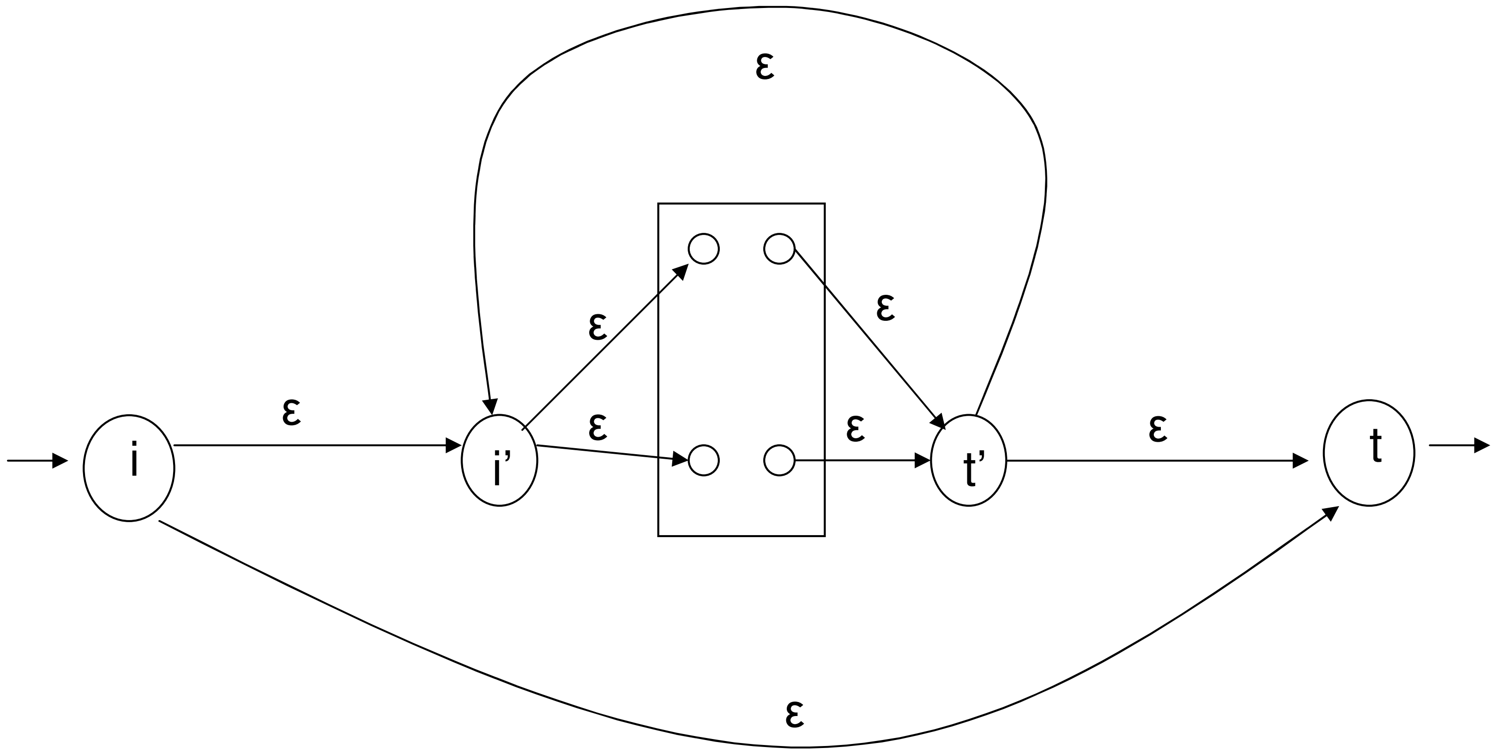
Concatenamento di due e.r.



Unione di due e.r.

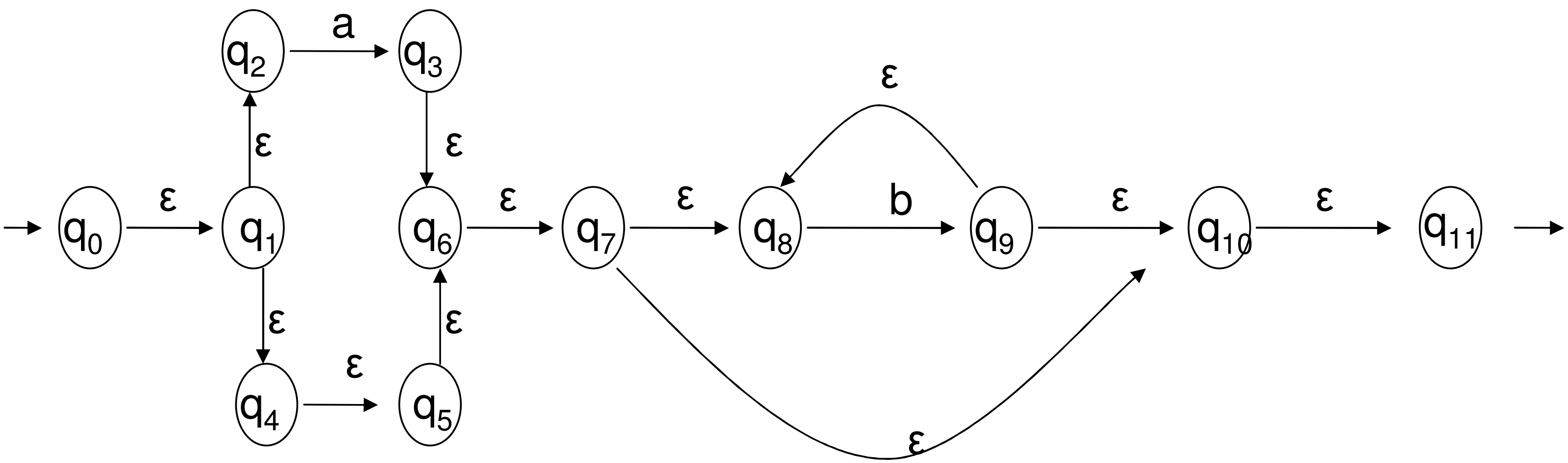


Stella di una e.r.



ESEMPIO

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



Esistono versioni dell'algoritmo che evitano la creazione di stati ridondanti (q_0 e q_{11}) o che eliminano le mosse spontanee.

ALGORITMO DI GLUSHKOV, Mc NAUGHTON e YAMADA (GMY)

Un altro metodo classico che costruisce un automa i cui stati sono in corrispondenza diretta con i caratteri terminali della e.r.

Sfrutta proprietà dei LINGUAGGI LOCALMENTE TESTABILI (LOC)

- i LOC sono ***sotto***famiglia dei linguaggi regolari ($LOC \subset REG$, $LOC \neq REG$)
- è facile
    - data e.r. che definisce un linguaggio $\in LOC$, costruire il suo automa riconoscitore
    - data una generica e.r. *e* (non necessariamente di un $L \in LOC$)
        - ricavare da questa l'e.r. e' di un $L' \in LOC$
        - dal cui automa riconoscitore si può ricavare il riconoscitore di L

DEFINIZIONE: dato un linguaggio L di alfabeto Σ

insieme degli inizi $Ini(L) = \{a \in \Sigma \mid a\Sigma^* \cap L \neq \emptyset\}$

insieme delle fini $Fin(L) = \{a \in \Sigma \mid \Sigma^* a \cap L \neq \emptyset\}$

insieme dei digrammi $Dig(L) = \{x \in \Sigma^2 \mid \Sigma^* x \Sigma^* \cap L \neq \emptyset\}$

il suo complemento $\overline{Dig}(L) = \Sigma^2 \setminus Dig(L)$

In un linguaggio $L_1 \in$ LOC i tre insiemi caratterizzano esattamente tutte le frasi, cioè qualunque stringa costruita nel rispetto di *Ini*, *Fin* e *Dig* appartiene al linguaggio.

$$L_1 \equiv \{x \mid Ini(x) \in Ini(L_1) \wedge Fin(x) \in Fin(L_1) \wedge Dig(x) \subseteq Dig(L_1)\} \quad (1)$$

o in modo equivalente:

$$L_1 \equiv \{x \mid Ini(x) \in Ini(L_1) \wedge Fin(x) \in Fin(L_1)\} \setminus \Sigma^* \overline{Dig}(L_1) \Sigma^* \quad (2)$$

ESEMPIO di linguaggio localmente testabile: $L_1 = (abc)^+$

$Ini(L_1) = \{a\}$ $Fin(L_1)=\{c\}$ $Dig(L_1)=\{ab, bc, ca\}$

NB: per ogni linguaggio (anche non regolare, purché non contenente ε) vale comunque la proprietà 1 sostituendo l'identità con l'inclusione. perchè, banalmente`, ogni frase

- inizia (risp. termina) con un carattere di *Ini* (risp. di *Fin*)
- i suoi digrammi sono inclusi in quelli del linguaggio.

Ma $L \notin LOC$ se esiste stringa $x \notin L$ tale che $Ini(x) \in Ini(L)$, $Fin(x) \in Fin(L)$ e $Dig(x) \subseteq Dig(L)$ (NB: questo è un semplice criterio per mostrare che un linguaggio NON è locale

ESEMPIO: linguaggio regolare non locale
L_2 <u>è strettamente contenuto</u> nell'insieme delle stringhe che iniziano e terminano con *b* e non contengono il digramma *bb*: Infatti L_2 non contiene stringhe di lunghezza dispari né stringhe con *b* circondate da *a* (e.g. *baabaabaab*).

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

L_1 ed L_2 regolari, L_1 è locale ma L_2 non è locale

Quindi la classe LOC dei linguaggi locali e` ***strettamente*** inclusa in REG

IL RICONOSCITORE DI UN LINGUAGGIO LOCALE È SEMPLICISSIMO:
scandendo la stringa, controlla che:

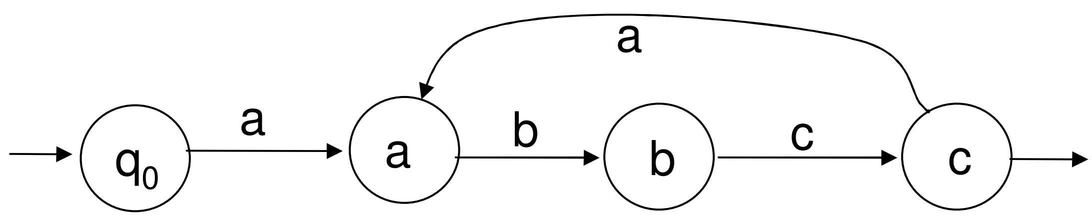
* il primo carattere stia in *Ini*
* ogni coppia di caratteri adiacenti stia in *Dig*
* l'ultimo carattere stia in *Fin*

Si può effettuare il controllo facendo scorrere una finestra mobile, larga due posizioni, da sinistra a destra sulla stringa, operazione attuabile con automa finito deterministico molto semplice.

Costruzione del riconoscitore di un linguaggio $L \in LOC$ a partire da insiemi Ini, Fin, Dig

- esiste un solo stato iniziale, q_0
- insieme degli stati $Q = \Sigma \cup \{q_0\}$ (tutti stati eccetto iniziale etichettati con un simbolo di Σ)
- stati finali F = Fin; se $\varepsilon \in L$ allora F include anche q_0
- funzione di transizione δ: $\forall a \in Ini\ \delta(q_0,a)=a$; $\forall xy \in Dig\ \delta(x,y)=y$

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



OSSERVAZIONI:

1) Gli stati non iniziali sono in corrispondenza biunivoca con l'alfabeto (quindi non ce ne sono più di $|\Sigma|$)
2) nessuna transizione verso lo stato iniziale q_0
3) tutte le frecce etichettate con la stessa lettera *a* entrano nello stesso stato, quello etichettato con *a* (quindi sul grafo delle transizioni le etichettature sugli archi e sulle trasizioni sono ridondanti – si può togliere una dei due)

CALCOLO DEGLI INSIEMI Ini, Fin, Dig DA UN'ESPRESSIONE REGOLARE

Diamo algoritmo sotto forma di funzioni ricorsive, definite per casi

Introduciamo preliminarmente predicato

$\text{Null}: ER \to \text{Boolean};$ Null(e) se e solo se ER e è annullabile

$\text{Null}(\varnothing) = \text{false}$

$\text{Null}(\varepsilon) = \text{true}$

$\text{Null}(a) = \text{false} \ \forall a \in \Sigma$

$\text{Null}(e_1 \mid e_2) = \text{Null}(e_1) \text{ or } \text{Null}(e_2)$

$\text{Null}(e_1 \cdot e_2) = \text{Null}(e_1) \text{ and } \text{Null}(e_2)$

$\text{Null}(e^*) = \text{true}$

$\text{Null}(e^+) = \text{Null}(e)$

Esempi: $\text{Null}((a \mid b)^* ba) = \text{true and Null}(ba) = \text{false}$
$\text{Null}(a (b \mid c)^*) = \text{false and true} = \text{false}$

Ini: ER $\to 2^\Sigma$ Ini(e) è l'ínsieme degli inizi di L(e)

$\mathrm{Ini}(\emptyset) = \mathrm{Ini}(\varepsilon) = \emptyset$

$\mathrm{Ini}(a) = \{a\}\ \forall a \in \Sigma$

$\mathrm{Ini}(e_1 \mid e_2) = \mathrm{Ini}(e_1) \cup \mathrm{Ini}(e_2)$

$\mathrm{Ini}(e_1 \cdot e_2) =$ if $\mathrm{Null}(e_1)$ then $\mathrm{Ini}(e_1) \cup \mathrm{Ini}(e_2)$ else $\mathrm{Ini}(e_1)$

$\mathrm{Ini}(e^*) = \mathrm{Ini}(e^+) = \mathrm{Ini}(e)$

Esempi: $\mathrm{Ini}(\ a\ (b \mid c)^*\ ) = \{a\}$ $\mathrm{Ini}((b \mid c)^*) = \{b, c\}$ $\mathrm{Ini}(\ (b \mid c)^*\ a\ ) = \{a, b, c\}$

Fin: ER $\to 2^\Sigma$ Fin(e) è l'insieme delle fini di L(e)

$\mathrm{Fin}(\emptyset) = \mathrm{Fin}(\varepsilon) = \emptyset$

$\mathrm{Fin}(a) = \{a\}\ \forall a \in \Sigma$

$\mathrm{Fin}(e_1 \mid e_2) = \mathrm{Fin}(e_1) \cup \mathrm{Fin}(e_2)$

$\mathrm{Fin}(e_1 \cdot e_2) =$ if $\mathrm{Null}(e_2)$ then $\mathrm{Fin}(e_1) \cup \mathrm{Fin}(e_2)$ else $\mathrm{Fin}(e_2)$

$\mathrm{Fin}(e^*) = \mathrm{Fin}(e^+) = \mathrm{Fin}(e)$

Esempi: $\mathrm{Fin}(\ (b \mid c)^*\ ) = \{b, c\}$ $\mathrm{Fin}(\ a\ (b \mid c)^*\ ) = \{a, b, c\}$

Dig: ER $\rightarrow \Sigma^2$ Dig(e) è l'insieme dei digrammi di L(e)

$\mathrm{Dig}(\varnothing) = \mathrm{Dig}(\varepsilon) = \varnothing$

$\mathrm{Dig}(a) = \varnothing \ \forall a \in \Sigma$

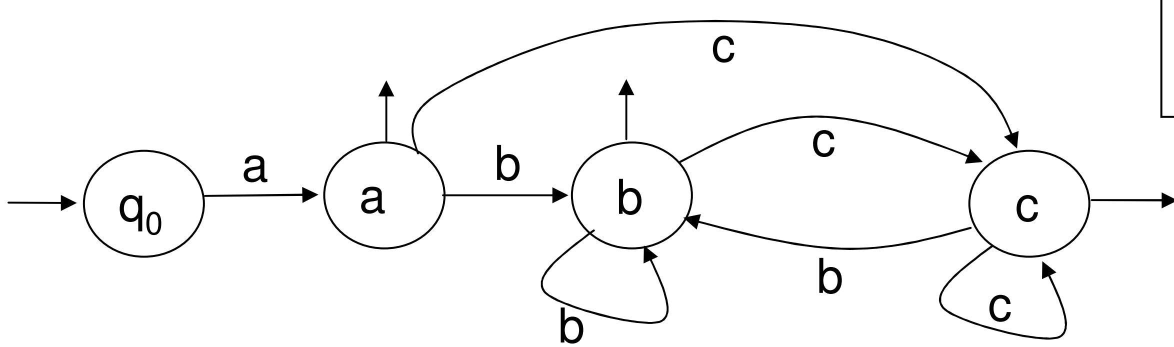
$\mathrm{Dig}(e_1 \mid e_2) = \mathrm{Dig}(e_1) \cup \mathrm{Dig}(e_2)$

$\mathrm{Dig}(e_1 \cdot e_2) = \mathrm{Dig}(e_1) \cup \mathrm{Fin}(e_1) \cdot \mathrm{Ini}(e_2) \cup \mathrm{Dig}(e_2)$

$\mathrm{Dig}(e^*) = \mathrm{Dig}(e^+) = \mathrm{Dig}(e) \cup \mathrm{Fin}(e) \cdot \mathrm{Ini}(e)$

Esempi: Dig(a (b | c)*) = {ab, ac} ∪ {bb, bc, cb, cc}

ESEMPIO – Riconoscitore dell'e.r. (locale) *a(bUc)**

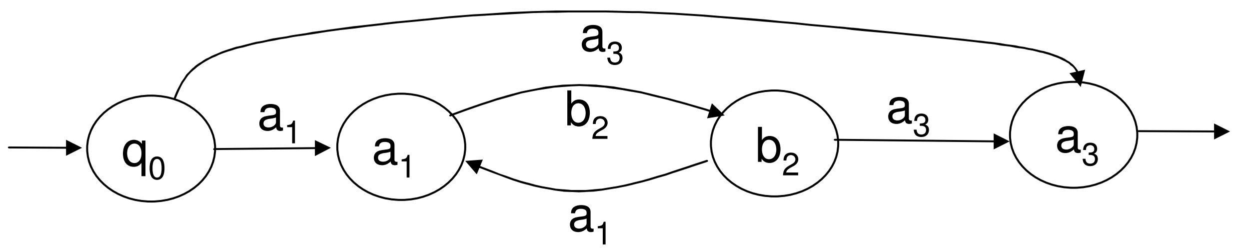


Ini={a}
Did={ab, ac, bb, bc, cb, cc}
Fin={a, b, c}

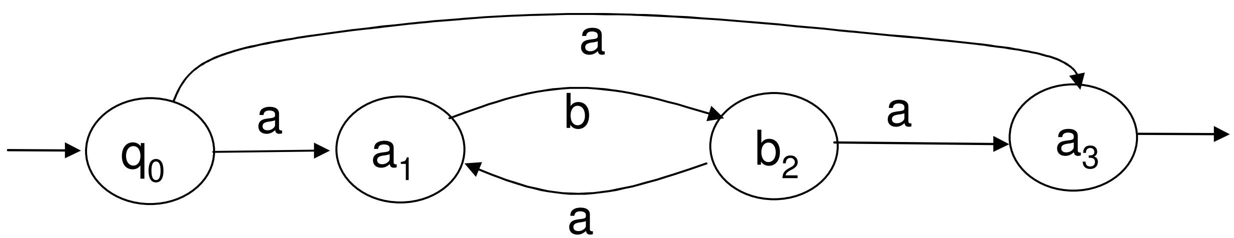
Avevamo annunciato una “ricetta” per ricavare il riconoscitore da una e.r.

data una generica e.r. *e* (non necessariamente di un L$\in$LOC)
ricavare da questa l’e.r. e’ di un L’$\in$LOC
dal cui automa riconoscitore si può ricavare il riconoscitore di L

Come si ottiene *e’*? Semplice: è la versione ***numerata*** di *e*
(vedremo poi che e’ definisce sempre un linguaggio locale)
es. $(ab)^*a$ diventa $(a_1b_2)^*a_3$ Ini=$\{a_1,a_3\}$ Dig=$\{a_1b_2, b_2a_1, b_2a_3\}$ Fin=$\{a_3\}$



Come si ottiene l’automa per la *e* di partenza ? Semplice: si toglie la numerazione
si perde informazione comunque inutile per il riconoscimento di L(*e*)
NB: tolta la numerazione, l’automa può risultare nondeterministico (cfr. esempio)



Abbiamo fatto una tacita assunzione, che va veriicata:
Data una qss. ER *e*, la versione numerata *e'* definisce un linguaggio locale
E' effettivamente verificata, grazie alla seguente proprietà:

dati due linguaggi $L', L'' \in LOC$, con $\Sigma' \cap \Sigma'' = \emptyset$, si ha $L'|L'', L' \cdot L'', L^* \in LOC$ (la composizione regolare di linguaggi locali con alfabeti disgiunti produce un linguaggio locale)

E ovviamente l'espressione numerata *e'* è formata da sottoespressioni aventi alfabeti tutti disgiunti

La località dei linguaggi composti da linguaggi locali si dimostra facilmente in modo costruttivo dando una procedura per ottenere riconoscitore del linguaggio risultante da quelli di partenza

Ovvia base di questa dimostrazione per induzione:

I linguaggi $\emptyset$, $\{\varepsilon\}$, $\{a\}$ per ogni $a \in \Sigma$, sono tutti locali

UNIONE:
- stato iniziale q_0: si fondono q_0' e q_0''
- stati finali: quelli dei due automi $F' \cup F''$

se la stringa vuota appartiene ad almeno uno dei due linguaggi q_0 è anche finale.

CONCATENAMENTO:

- stato iniziale: q'_0
- stati finali: se $\varepsilon \notin L''$

  gli stati finali F''

  altrimenti (se $\varepsilon \in L''$)

  gli stati finali F' uniti a quelli di F''
- mosse:
  - le mosse di L'
  - le mosse di L'' tranne quelle con origine in q''_0
  - le mosse

    $q' \xrightarrow{a} q''$ per ogni $q' \in F'$ e per ogni $q''_0 \xrightarrow{a} q''$ incluso in L'' (cioè $\forall a \in Ini(L'')$)

STELLA:

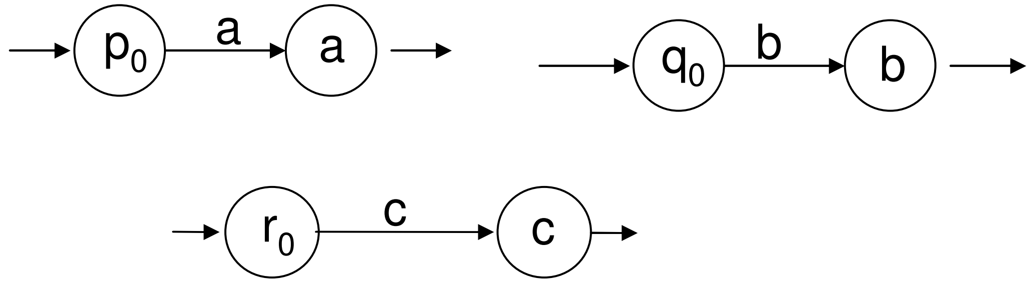
- si aggiunge q_0 agli stati finali
- per ogni stato finale q in F' si aggiunge

  $q \xrightarrow{a} r$ se l'automa aveva $q_0 \xrightarrow{a} r$ (cioè per ogni $a \in Ini(L)$)

ESEMPIO – (ab U c)*

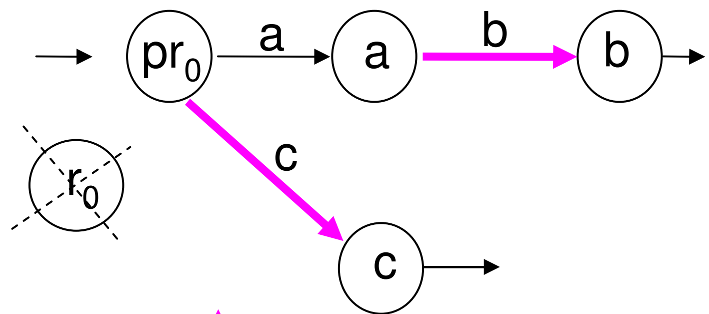
automi elementari

a,b,c



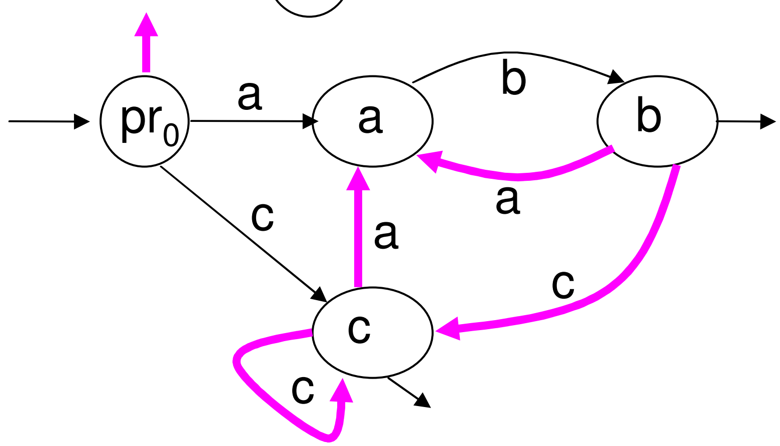
concatenamento e unione

ab U c



stella

(ab U c)*



ALGORITMO GMY (in sintesi) per ottenere riconoscitore da e.r. *e* :

1) Numera l'e.r. di partenza *e* per ottenere *e'* locale
2) Calcola per *e'* *Ini*, *Fin* e *Dig*
3) Costruisci il riconoscitore del linguaggio locale L(*e'*)
4) Cancella la numerazione dalle etichette degli archi del riconoscitore
5) Risultato: un automa indeterministico privo di mosse spontanee, avente tanti stati quanti sono i caratteri terminali della e.r. più uno

COSTRUZIONE DEL RICONOSCITORE DETERMINISTICO DI BERRY E SETHI

Per ottenere un automa deterministico potremmo applicare l'algoritmo delle parti finite all'automa costruito da GMY, ma conviene utilizzare un algoritmo diretto.

Sia e l'e.r. di partenza (di alfabeto Σ),
e' la versione numerata di *e* (di alfabeto Σ_N con i simboli numerati)

Si considera $e'\dashv$ *,* si definisce l'insieme dei séguiti, con riferimento ai soliti Ini, Fin, Dig

$\text{Seg}(a) = \{ b \mid ab \in \text{Dig}(e') \}$

$\dashv \in \text{Seg}(a)$ per ogni $a \in \text{Fin}(e')$

$\text{Seg}(\dashv) = \emptyset$

ALGORITMO BS

Uno stato è denotato da un sottoinsieme di $\Sigma_N \cup \dashv$.

L'algoritmo esamina ogni stato per costruire le mosse uscenti e gli stati di arrivo, applicando una regola simile a quella dell'algoritmo delle parti finite.

Lo stato viene anche marcato come visitato per evitare di riesaminarlo.

Lo stato inziale è *Ini(e' ┤)*.
Uno stato è fin. se contiene ┤
Inizialmente l'insieme degli stati Q contiene il solo stato iniziale.

$Q := \{ Ini(e' \dashv) \}$

while esiste in Q uno stato q non visitato

do

segna q come visitato

per ogni car. $b \in \Sigma$

do

$q' := \bigcup_{\forall \text{car.numerato } b' \in q} Seg(b')$

se q' non è vuoto né appartiene a Q,

aggiungilo come nuovo stato

non visitato ponendo:

$Q := Q \cup \{q'\}$

aggiungi la mossa $q \xrightarrow{b} q'$

end do

end do

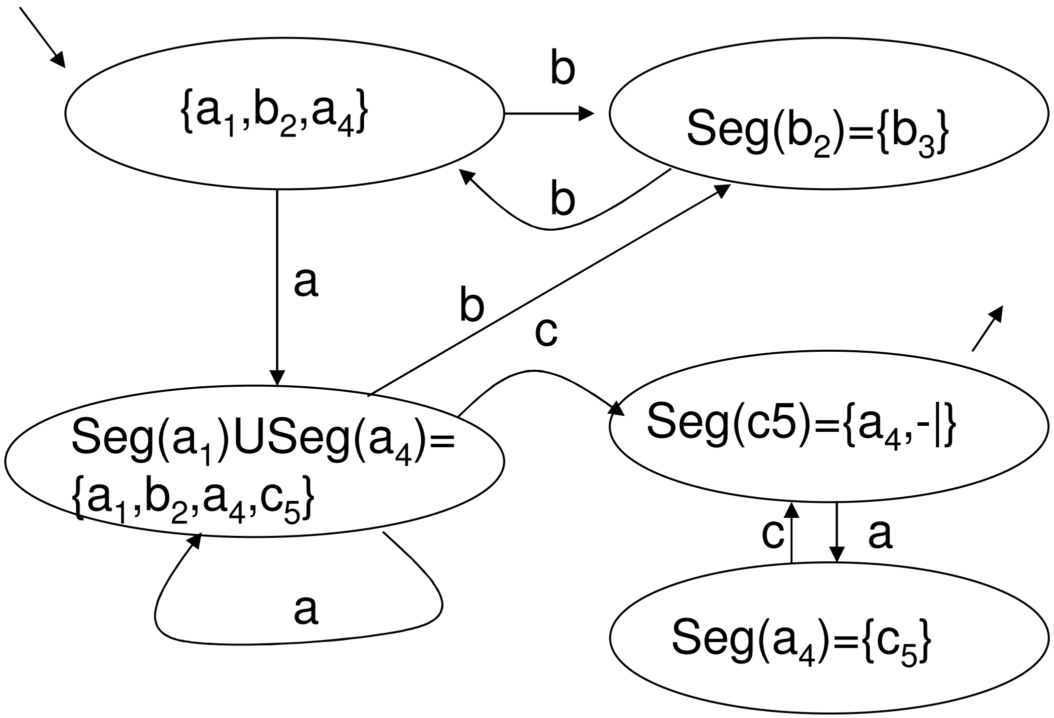
ESEMPIO

NOTA: i nodi di un automa costruito con BS riportano *"cosa ci si può aspettare"* ... non *"cos'è arrivato"*!

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

$$(a_1 \mid b_2 b_3)^*(a_4 c_5)^+ -\!|$$

$$Ini(e' -\!|) = \{a_1, b_2, a_4\}$$



Séguiti

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, -|

Gli automi costruiti con questi procedimenti sono deterministici ma, per via della numerazione, possono contenere più stati del necessario, e si possono poi minimizzare.

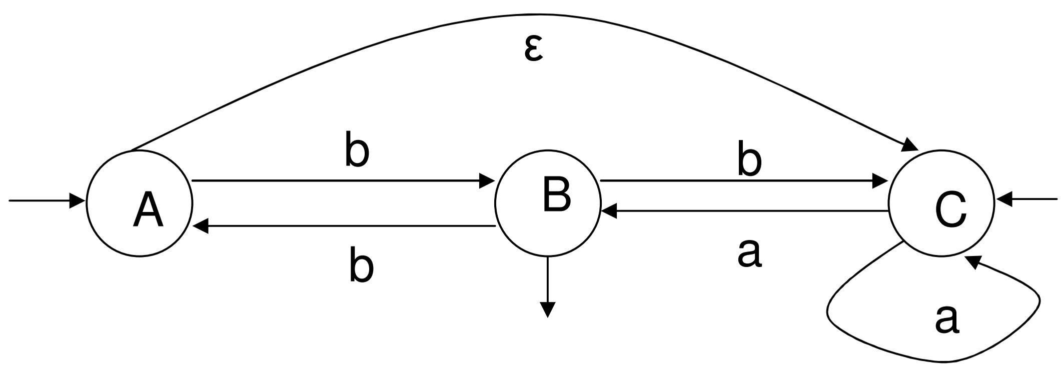
ALGORITMO BS PER LA DETERMINIZZAZIONE DI UN AUTOMA

L'algoritmo BS può essere usato in alternativa a quello delle parti finite per la determinizzazione di un automa anche contenente ε-archi

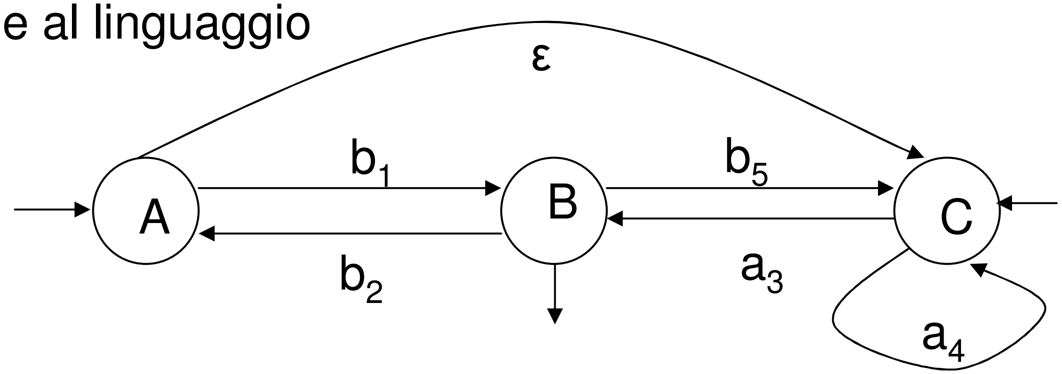
1) Si numerano in modo distinto le etichette degli archi non spontanei di N. L'automa numerato N' ottenuto riconosce un linguaggio locale
2) Si calcolano gli insiemi locali *Ini, Fin, Seguiti* per l'automa N' (con regole analoghe a quelle usate per una e.r.
3) Si applica la costruzione di BS, per ottenenre l'automa deterministico M.

Automa risultante è deterministico, anche se potrebbe non essere minimo

ESEMPIO



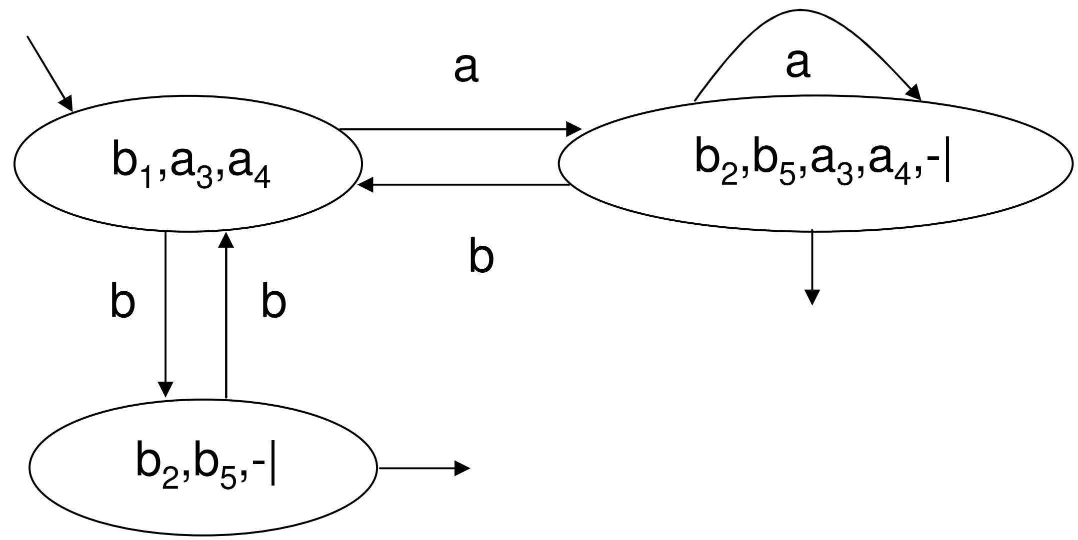
Il linguaggio è locale
ε non appartiene al linguaggio



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

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

	Séguiti
b_1	$b_2, b_5,$ -|
b_2	b_1, a_3, a_4
a_3	$b_2, b_5,$ -|
a_4	a_3, a_4
b_5	a_3, a_4



ESPRESSIONI REGOLARI CON COMPLEMENTO E INTERSEZIONE

Completiamo lo studio delle operazioni sui linguaggi regolari considerando il complemento, l’intersezione e la differenza insiemistica, lasciati in sospeso.
Questi operatori possono rendore più facile o più concisa l’espressione del linguaggio.

PROPRIETÀ - CHIUSURA DI REG. RISPETTO A COMPLEMENTO E INTERSEZIONE

$$\boxed{\begin{array}{l} \text{Se } L, L', L'' \in REG \\ \text{allora } \neg L \in REG \quad L' \cap L'' \in REG \end{array}}$$

1) COSTRUZIONE DEL RICONOSCITORE DETERMINISTICO DEL COMPLEMENTO

$$\boxed{\neg L = \Sigma^* \setminus L}$$

Possiamo supporre che il riconoscitore M di L sia deterministico, con stato iniziale q_0, stati Q, stati finali F e con δ funzione di transizione.

ALGORITMO: costruzione del riconoscitore deterministico $\overline{M}$ del complemento.

Estendiamo *M* con lo stato pozzo *p* e con le mosse che in esso cadono.

1. Crea un nuovo stato *p* non appartenente a *Q* (il pozzo). Gli stati dell'automa complemento sono *Q U {p}*.
2. La funzione di transizione è: $\longrightarrow$

$$\boxed{\begin{array}{l} a)\ \overline{\delta}(q,a) = \delta(q,a),\ \text{se } \delta(q,a) \text{ è definita} \\ b)\ \overline{\delta}(q,a) = p \text{ altrimenti} \\ c)\ \overline{\delta}(p,a) = p \text{ per ogni car. } a \in \Sigma \end{array}}$$

3. Si scambiano poi gli stati finali e non finali della macchina:

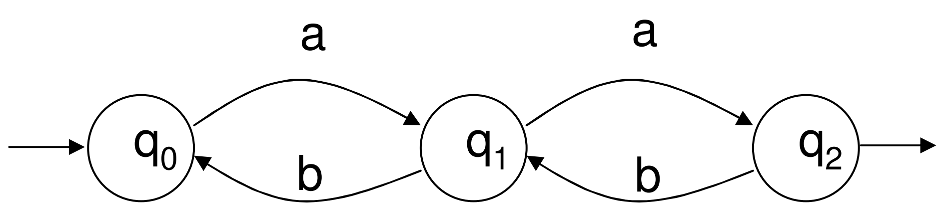
$$\boxed{\overline{F} = (Q \setminus F) \bigcup \{p\}}$$

Se un calcolo di M riconosce la stringa, il corrispondente calcolo di $\overline{M}$ termina in uno stato non finale. Inoltre se un calcolo di M non riconosce y, due sono i casi possibili: il calcolo termina in uno stato q non finale, o il calcolo termina nello stato pozzo p. In entrambi i casi il calcolo corrispondente di $\overline{M}$ termina in uno stato finale.

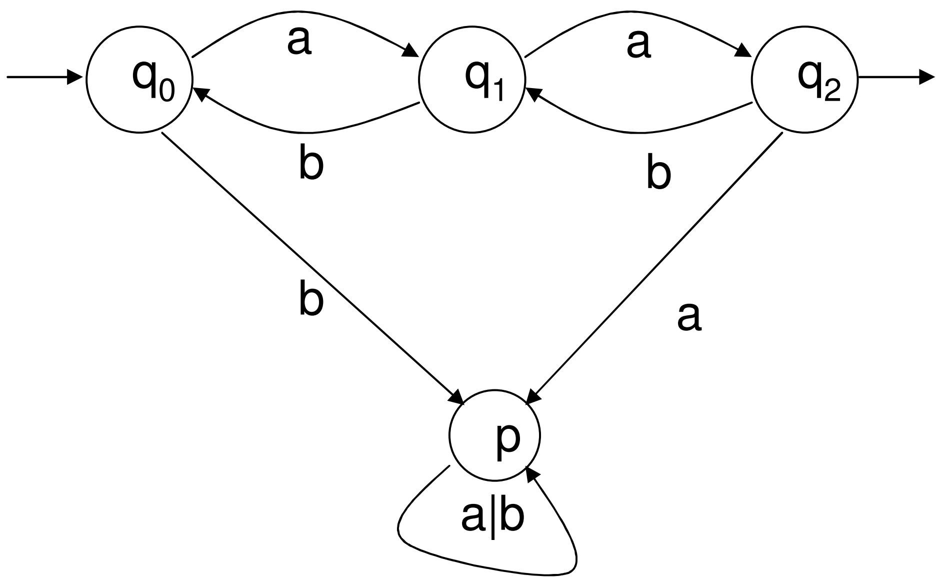
$$\boxed{\begin{array}{l} x \in L(M) \Rightarrow x \notin L(\overline{M}) \\ y \notin L(M) \Rightarrow y \in L(\overline{M}) \\ \text{quindi} \quad L(\overline{M}) = \Sigma^* \setminus L(M) \end{array}}$$

ESEMPIO: AUTOMA DEL COMPLEMENTO

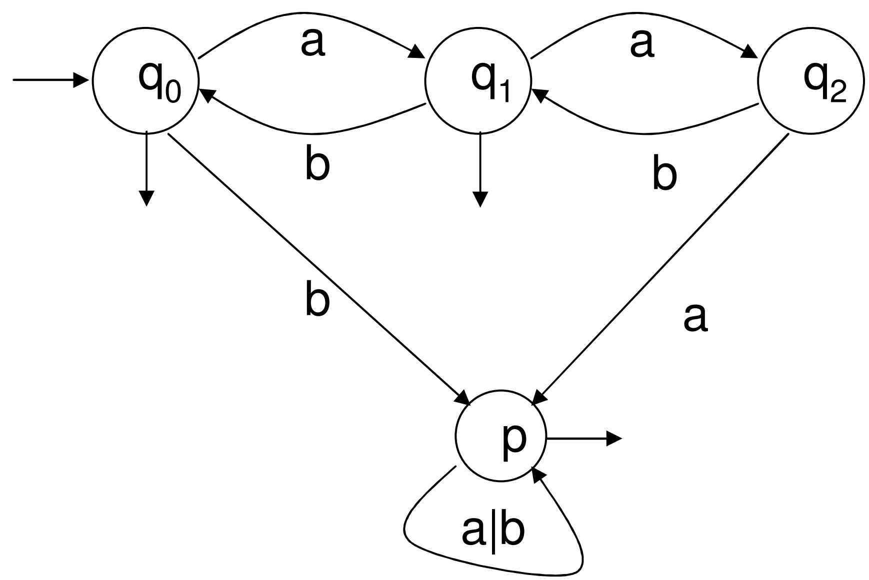
Automa originale:



Automa completato con lo stato pozzo:



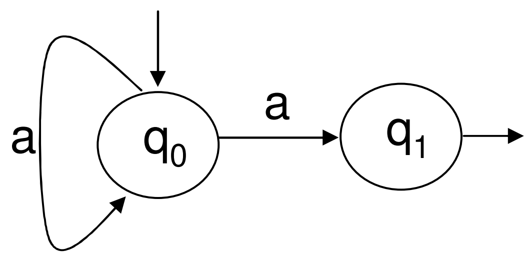
Automa complementato
(scambiati stati finali):



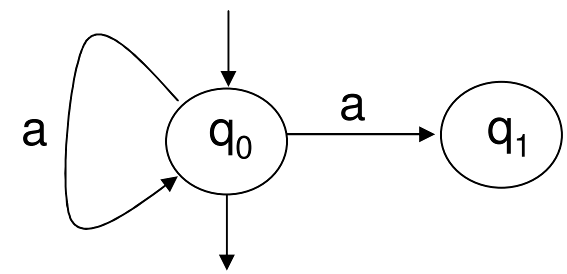
NB: essenziale che l'automa di partenza sia deterministico,
altrimenti il linguaggio accettato dall'automa costruito potrebbe non essere disgiunto da quello originale

Esempio:

$$\boxed{L \cap \neg L = \varnothing}$$



automa originale (nondeterministico)



pseudo-automa del complemento

Lo pseudo-automa del complemento accetta la stringa *a* appartenente al linguaggio originale.

OSSERVAZIONE: la costruzione può produrre automi con stati inutili, o comunque automi non minimi.

2) DIMOSTRIAMO CHE L'INTERSEZIONE DI DUE LINGUAGGI REGOLARI È REGOLARE

Trasformiamo con identità di De Morgan ...

$$\boxed{L' \cap L'' = \neg(\neg L' \cup \neg L'')}$$

l'unione di due linguaggi regolari è regolare.

COROLLARIO: anche la DIFFERENZA INSIEMISTICA DI DUE LINGUAGGI REGOLARI È REGOLARE, per l'identità:

$$\boxed{L' \setminus L'' = L' \cap \neg L''}$$

PRODOTTO (CARTESIANO) DI AUTOMI

Metodo molto comune della teoria degli automi che consiste nella simulazione di due automi mediante un automa detto il prodotto (cartesiano) di automi.
Ne vediamo un'applicazione per la costruzione del riconoscitore del linguaggio intersezione.

Per costruire il riconoscitore dell'intersezione potremmo sfruttare la dimostrazione della chiusura di REG rispetto all'intersezione (basata sull'identità di De Morgan $L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$) e quindi:

- costruire i riconoscitori deterministici dei due linguaggi
- costruire quelli dei loro complementi
- costruire la loro unione (con il metodo di Thomson)
- rendere l'automa ottenuto deterministico
- costruire infine l'automa del complemento

Esiste però un metodo più diretto.

IN MODO PIÙ DIRETTO, l’intersezione di due linguaggi può essere riconosciuta da UNA MACCHINA detta PRODOTTO (CARTESIANO) che simula il comportamento dei due automi.

Supponiamo che i due automi siano privi di mosse spontanee ma non necessariamente deterministici.

La macchina prodotto M ha come insieme degli stati il prodotto cartesiano degli stati dei due automi M’ e M’’. Uno stato contiene una coppia <q’, q’’>, la prima (seconda) componente è uno stato della prima (seconda) macchina.

In tale stato si definisce la mossa:

$$\boxed{< q', q'' > \xrightarrow{a} < r', r'' > \text{ se, e solo se } q' \xrightarrow{a} r' \wedge q'' \xrightarrow{a} r''}$$

M ha una mossa se, e solo se, la sua proiezione sulla prima (risp. seconda) componente è una mossa di M’ (risp. di M’’).

Gli stati iniziali I di M sono il prodotto cartesiano I = I’ x I’’ degli stati iniziali delle macchine componenti e quelli finali sono il prodotto degli stati finali, F = F’ x F’’.

GIUSTIFICAZIONE DELLA COSTRUZIONE:

1) Una stringa x appartenente all’intersezione è accettata da due calcoli di M’ e M’’, dunque anche da un calcolo di M che passa attraverso le coppie di stati rispettivamente visitate dai due calcoli.

2) Se x non appartiene all’intersezione, almeno uno dei due calcoli di M’ e M’’ non raggiunge uno stato finale, dunque neanche il calcolo di M raggiunge uno stato finale.

NOTE:

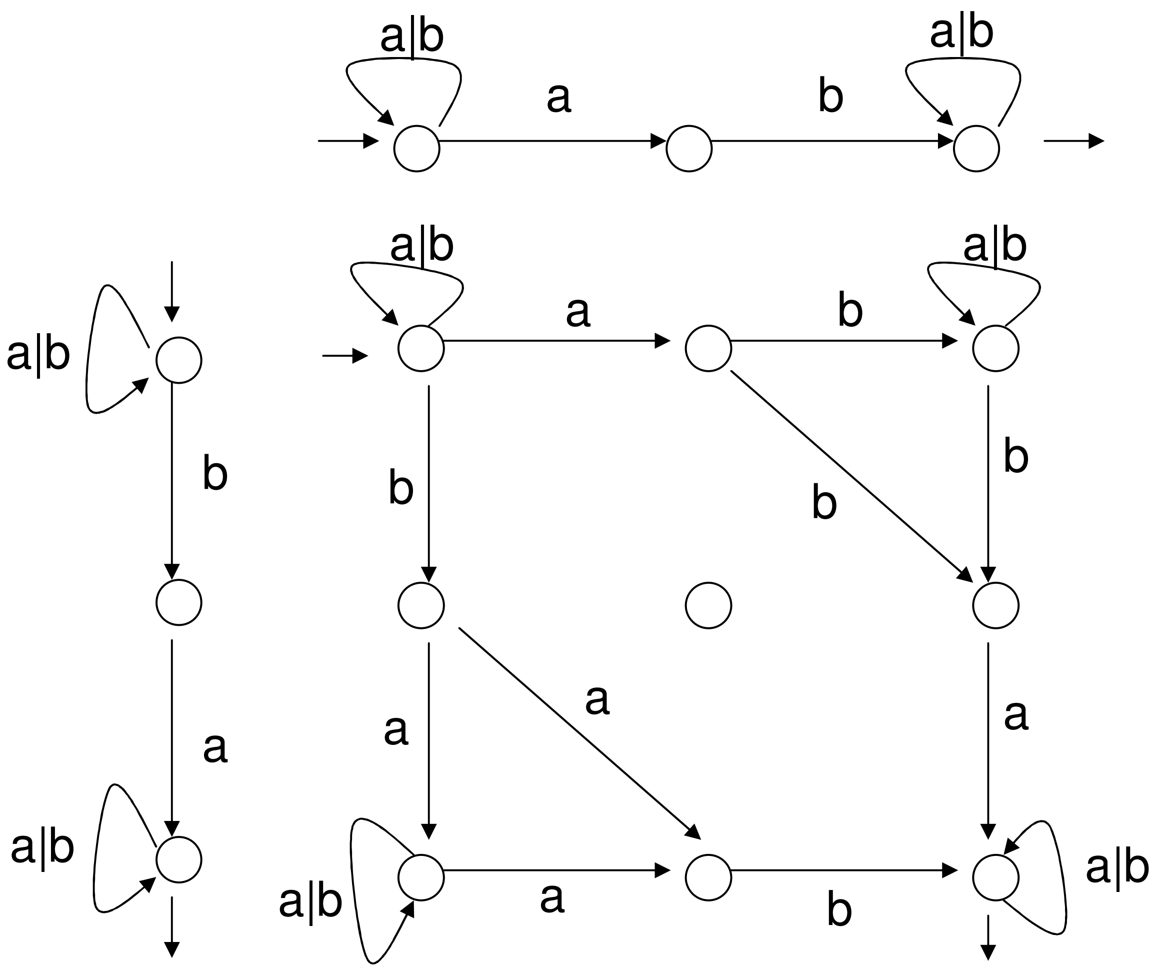
1) Il riconoscitore dell’intersezione può essere simulato eseguendo in parallelo i due automi con l’avvertenza che, se la mossa non è possibile per uno di essi, la stringa venga rifiutata. In tale modo si evita il costo della costruzione della macchina prodotto

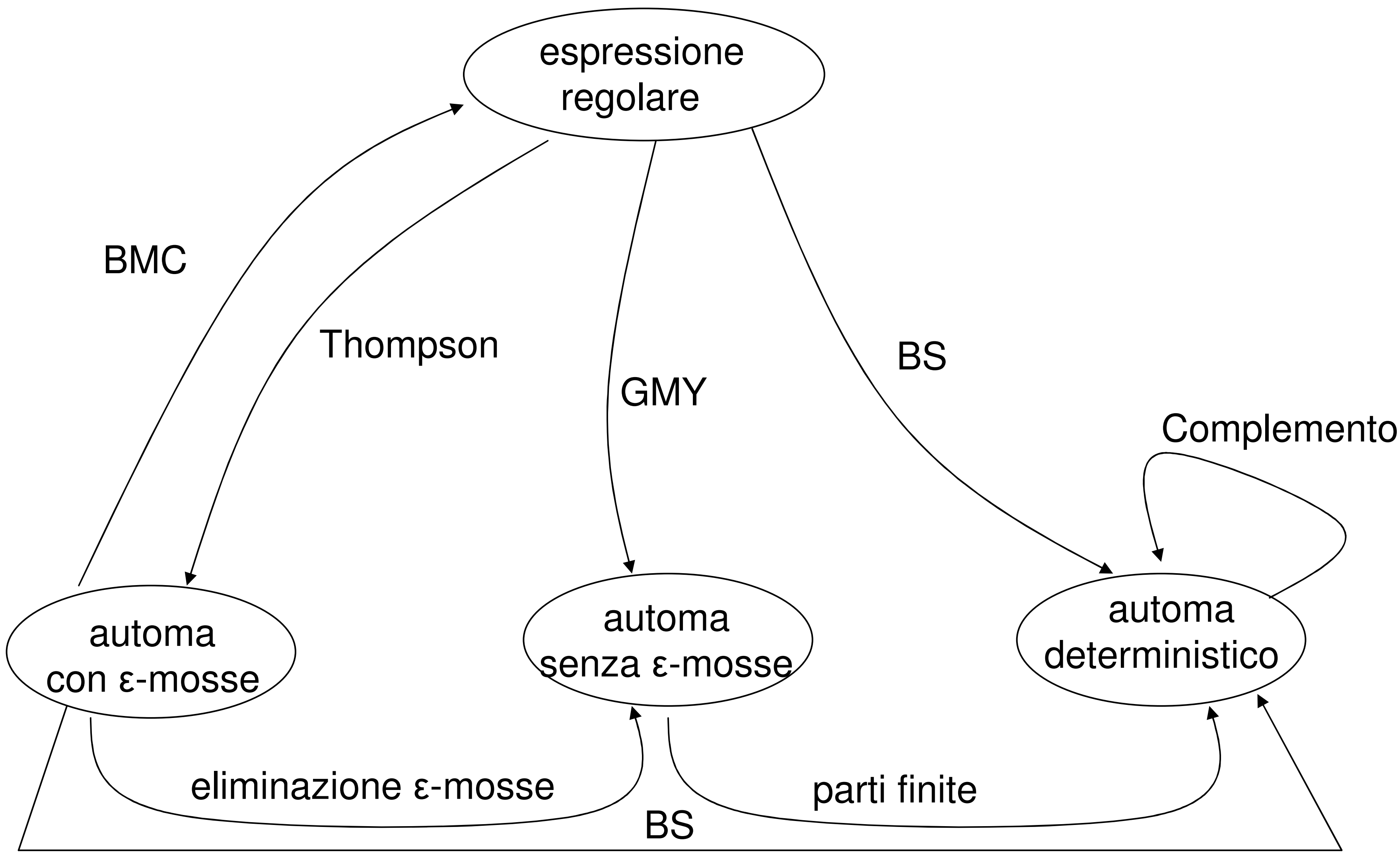
2) Il metodo del prodotto cartesiano può essere applicato ad altre operazioni diverse dall’intersezione (nel caso dell’unione si potrebbe modificare la macchina prodotto in modo che essa riconosca una stringa se almeno una delle due macchine componenti la riconosce.

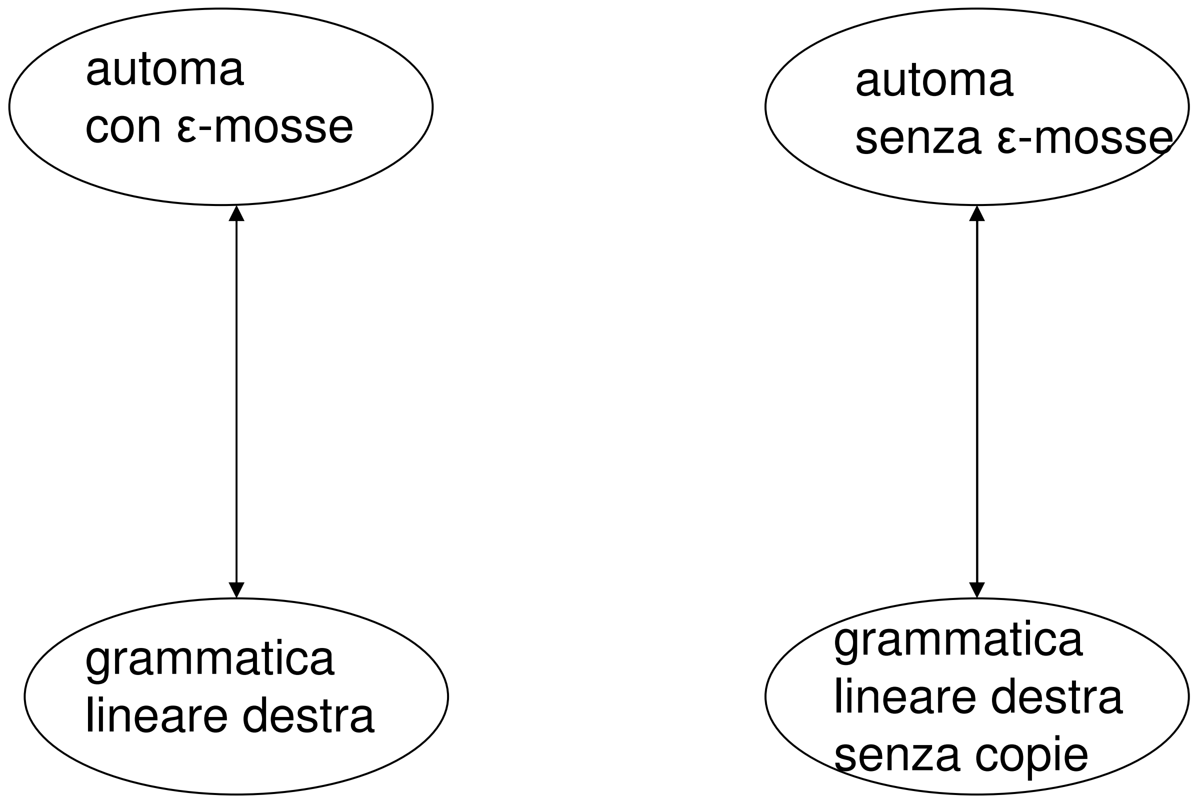
ESEMPIO – Intersezione e macchina prodotto per i due linguaggi che contengono, rispettivamente, la stringa *ab* e la stringa *ba*

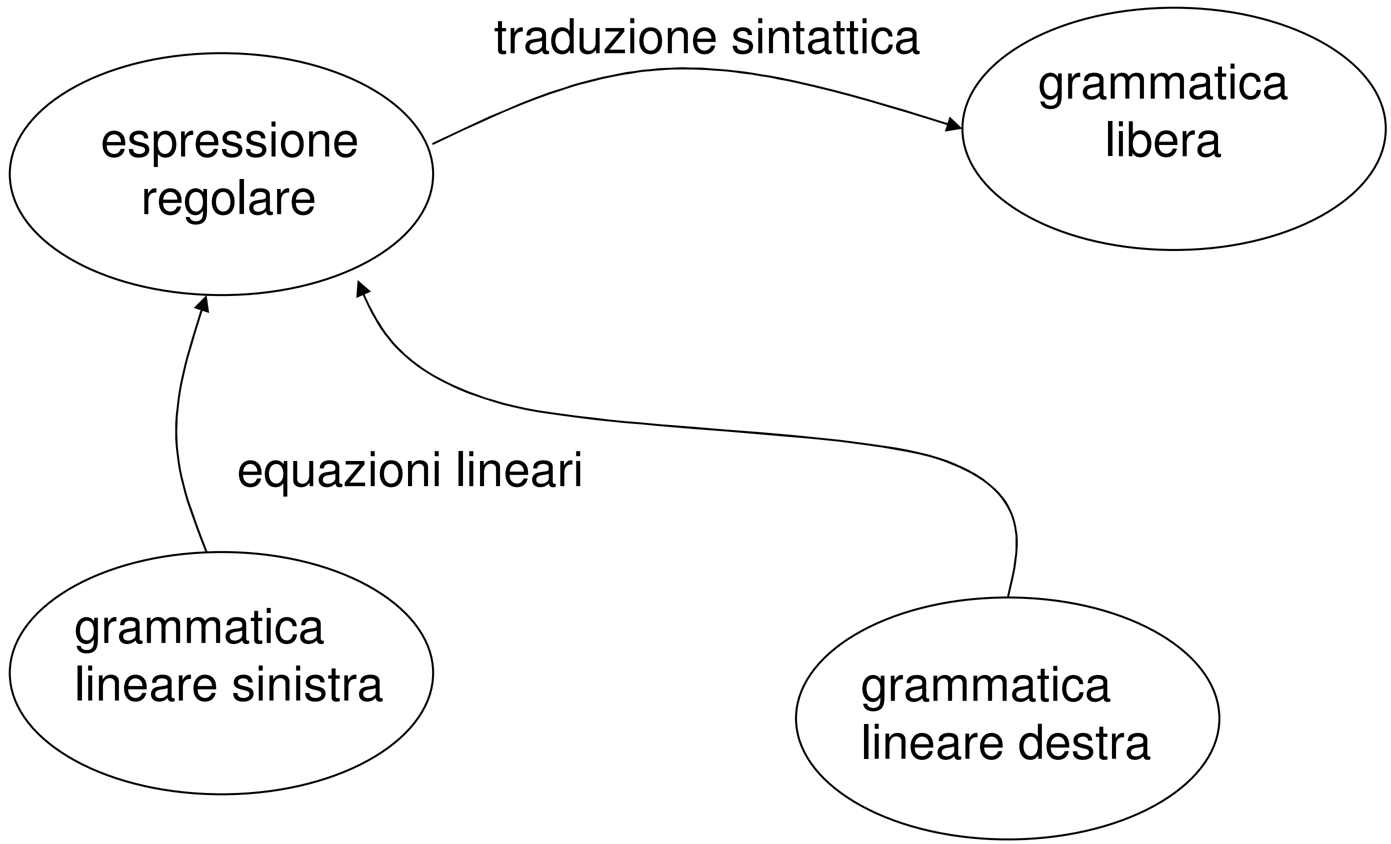
$$L' = (a \mid b)^{*} ab (a \mid b)^{*}$$

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




espressione regolare
BMC
Thompson
GMY
BS
Complemento
automa con ε-mosse
automa senza ε-mosse
automa deterministico
eliminazione ε-mosse
parti finite
BS


automa
con ε-mosse
automa
senza ε-mosse
grammatica
lineare destra
grammatica
lineare destra
senza copie


traduzione sintattica
espressione regolare
grammatica libera
equazioni lineari
grammatica lineare sinistra
grammatica lineare destra