

Automi finiti e riconoscimento dei linguaggi regolari

3.1 Introduzione

L'impiego delle grammatiche e delle espressioni regolari è molto diffuso nella scrittura dei manuali e dei documenti di standardizzazione dei linguaggi. Ma per progettare un compilatore, cioè un algoritmo che verifica la correttezza d'una frase e la traduce, è necessario un approccio più algoritmico, esposto in questo capitolo per i linguaggi regolari e nel prossimo per quelli liberi.

Il problema di riconoscere se un testo appartiene a un certo linguaggio formale, prima di tradurlo o elaborarlo con modalità dipendenti dall'applicazione, sorge molto frequentemente. Un compilatore analizza il programma sorgente per controllare che sia valido; un programma di videoscrittura verifica che le parole digitate siano ortografiche e soddisfino la sintassi, e un'interfaccia interattiva a finestre, prima di accettare un dato inserito dall'utente ne controlla la validità.

Tale controllo, essenziale nel trattamento dei testi, è svolto da un *algoritmo di riconoscimento*, così detto perché riconosce se una stringa è corretta rispetto a una definizione formale di riferimento. Le procedure di riconoscimento si specificano e progettano impiegando dei modelli minimalisti, detti *macchine astratte* o *automi*. Così facendo si mettono in risalto le peculiarità delle varie classi di riconoscitori, e dunque delle famiglie di linguaggi riconosciuti, senza appesantire l'esposizione con inutili particolari realizzativi dipendenti dalle scelte tecniche di progetto dell'algoritmo.

Nel capitolo, dopo un cenno agli automi matematici più generali, si presentano gli automi con memoria finita, che riconoscono i linguaggi definiti da espressioni regolari. Molte esigenze di ricerca delle parole in un testo o di estrazione degli elementi lessicali sono risolte con tali modelli.

Il primo modello di automa è quello deterministico, di cui si espongono i metodi basilari di ripulitura e di minimizzazione.

Poi si introduce in modo motivato il modello non deterministico, e se ne mostra la corrispondenza con le grammatiche del tipo unilineare.

Si presentano poi gli algoritmi per effettuare la conversione tra automa rico-

noscitore e espressione regolare del linguaggio riconosciuto, giungendo così al punto di arrivo concettuale, l'identità dei due modelli. Ciò facendo si introduce la sottofamiglia dei linguaggi locali, riconoscibili con dei test locali sulla stringa.

Segue la considerazione degli operatori di complemento e intersezione delle espressioni regolari e del metodo di composizione di automi mediante prodotto cartesiano.

Il capitolo termina con il riepilogo delle relazioni esistenti tra automi finiti, espressioni regolari e grammatiche.

Gli automi finiti sono fondamentali per la compilazione e riappariranno più volte nei prossimi capitoli del libro, in particolare come traduttori, ossia macchine per tradurre un testo nel capitolo 5. Nello stesso capitolo saranno viste altre applicazioni degli automi finiti nell'analisi statica del flusso d'un programma da verificare o da ottimizzare.

3.2 Algoritmi di riconoscimento e automi

Gli *algoritmi di riconoscimento* (o di decisione) sono spesso considerati dagli studiosi della teoria della complessità del calcolo, e non solo per i problemi dei linguaggi artificiali. Ad es. il celebre problema di decidere se due grafi sono isomorfi si riformula in questi termini come problema di riconoscimento: il dominio del problema è costituito da coppie di grafi e l'algoritmo deve emettere la risposta *sì/no*, a seconda che i grafi della coppia siano o meno isomorfi. Questo è sempre il codominio d'un algoritmo di riconoscimento, anche se, a seconda del problema, il dominio cambia.

Passando ai linguaggi, il dominio è un insieme di stringhe d'un alfabeto Σ . L'applicazione dell'algoritmo di riconoscimento α a una stringa data x si può indicare come $\alpha(x)$. Si dice che la stringa x è (risp. non è) *riconosciuta* o *accettata* se $\alpha(x) = \text{sì}$ (risp. $\alpha(x) = \text{no}$).

Il linguaggio riconosciuto, $L(\alpha)$, è allora l'insieme delle stringhe per cui:

$$L(\alpha) = \{x \in \Sigma^* \mid \alpha(x) = \text{sì}\}$$

Può succedere che per qualche stringa scorretta $x \in (\Sigma^* \setminus L(\alpha))$ l'algoritmo non termini, ossia $\alpha(x)$ non sia definita: in questo caso si dice che il problema di decidere l'appartenenza di x a L è semidecidibile, o anche che il linguaggio L è ricorsivamente enumerabile. Se un linguaggio fosse semidecidibile, non si potrebbe garantire che il compilatore non entri in ciclo quando analizza certe stringhe sorgente.

In pratica queste preoccupazioni, pur centrali per la teoria della computabilità,¹ non sono rilevanti per l'ingegneria del linguaggio, dove si evitano deliberatamente le famiglie di linguaggi semidecidibili. Tuttavia due problemi

¹Il lettore interessato a un approfondimento può consultare un testo di informatica teorica (ad es. Bovet e Crescenzi [11], Floyd e Beugel [19], Hopcroft e Ullman [28], Mandrioli e Ghezzi [32], Mc Naughton [33]).

indecidibili, diversi dal riconoscimento, sono stati incontrati: quello di verificare se due grammatiche sono equivalenti e di decidere se una grammatica è ambigua.

Nella pratica il riconoscimento è soltanto la prima parte del lavoro della compilazione. Nel capitolo 5, scostandosi dai puri problemi di riconoscimento, si studierà la traduzione d'un linguaggio in un altro. Il codominio d'una traduzione è più ricco di $\{si, no\}$, essendo esso stesso un linguaggio.

È noto che gli algoritmi di risoluzione dei problemi decidibili possono essere classificati in base alla loro complessità, misurata dalle risorse di calcolo, cioè della quantità di tempo (oppure di memoria), che richiedono. Quasi tutti i problemi di interesse per la compilazione hanno delle soluzioni di complessità bassa, lineare o al peggio polinomiale rispetto alle dimensioni del problema da risolvere.

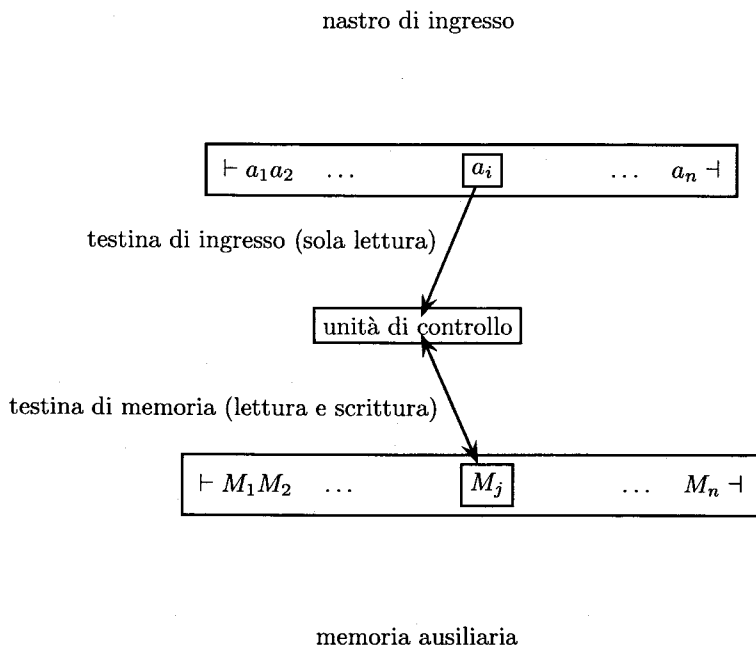
Quando si studia la complessità di calcolo, invece di misurare il tempo di esecuzione si preferisce contare il numero di passi compiuti dall'algoritmo. La ragione è che così facendo si hanno delle misure che non dipendono dalla velocità dell'elaboratore, ma soltanto dalla natura dell'algoritmo. I passi elementari dell'algoritmo sono operazioni più o meno ricche, a seconda del modello di calcolatore (in senso astratto) cui si pensa: a un estremo, un passo è un'istruzione d'un linguaggio programmatico; all'altro estremo un passo è l'operazione d'un automa o macchina astratta, capace solo di leggere e scrivere un simbolo. L'uso d'una macchina astratta è preferito nella teoria e nell'ingegneria del linguaggio, tanto da caratterizzarle rispetto a altri campi dell'informatica. In questo campo è consuetudine presentare gli algoritmi di riconoscimento e traduzione sotto la veste di automi. I vantaggi sono duplici: evidenziare la correlazione tra le varie famiglie di automi riconoscitori e le corrispondenti famiglie di linguaggi e di grammatiche; e evitare un riferimento prematuro agli aspetti realizzativi del software. Ciò facendo non si perdono di vista gli sviluppi applicativi, perché è facile dare le indicazioni essenziali per trasformare un automa in un programma.

3.2.1 Automa generale

Un automa o macchina astratta è un dispositivo di calcolo ideale, dotato d'un insieme ridotto di operazioni elementari assai semplici. La varietà di questi dispositivi si è continuamente arricchita nel corso della storia della teoria dei linguaggi e degli automi, a partire dai classici studi di A. Turing degli anni 1930 sull'omonima macchina di calcolo. Nel libro si studieranno soltanto i pochi tipi di automi impiegati per l'elaborazione dei linguaggi tecnici.² Lo schema d'un automa riconoscitore, nella sua forma più generale, comprende

²Per approfondire la teoria generale degli automi si rinvia il lettore a Salomaa [43], Hopcroft e Ullman [27, 28], Harrison [24]; per gli automi finiti un riferimento specifico è Sakarovitch [42].

tre parti: il nastro d'ingresso, l'unità di controllo, e la memoria ausiliaria:



L'unità di controllo ha una memoria limitata (si dice che ha un numero finito di stati), mentre la memoria ausiliaria ha in generale capacità illimitata. Sul nastro superiore sta scritta la stringa di ingresso (o *sorgente*), che può essere letta ma non modificata; esso è suddiviso in caselle, ciascuna contenente un simbolo dell'alfabeto terminale. Talvolta si impone che la stringa sia delimitata a sinistra e a destra da due caratteri riservati \vdash (marca d'inizio) e \vdash (marca di fine o terminatore). Anche la memoria ausiliaria può essere vista come un nastro, contenente simboli d'un alfabeto di memoria.

L'automa è un sistema che opera in istanti discreti svolgendo diverse azioni: leggere il *carattere* a_i *corrente*, spostare la testina di ingresso sul nastro a destra o sinistra d'una posizione, leggere il simbolo corrente M_j dalla memoria, e scrivere un simbolo al suo posto, spostare la testina di memoria, e cambiare lo stato della unità di controllo.

L'automa esamina la stringa sorgente compiendo una serie di mosse; ogni mossa dipende dai simboli presenti sotto le due testine e dallo stato del controllo. Una mossa può avere i seguenti effetti:

- spostamento della testina di ingresso d'una posizione a destra o a sinistra;

- scrittura d'un simbolo al posto del simbolo corrente nella memoria e spostamento della testina di memoria (d'una posizione a destra o sinistra);
- cambiamento di stato dell'unità di controllo.

Alcune di queste azioni possono mancare.

L'automa è *monodirezionale* se la testina d'ingresso si può spostare soltanto da sinistra verso destra: questo è il solo caso considerato, perché meglio risponde all'esigenza di tradurre un testo con una sola scansione.

Il comportamento futuro dell'automa è definito da tre componenti, costituenti la *configurazione* (istantanea): il suffisso non ancora letto della stringa sorgente, situato a destra della testina d'ingresso; il contenuto della memoria ausiliaria e la posizione della testina di memoria; lo stato della unità di controllo. L'automa è all'origine posto nella *configurazione iniziale*: la testina d'ingresso sta sul simbolo a_1 che segue la marca d'inizio, l'unità di controllo è nello stato iniziale, e la memoria contiene l'informazione iniziale (solitamente un particolare simbolo).

Attraverso una serie di mosse (calcolo o computazione) la configurazione dell'automa evolve. Il cambiamento è *deterministico* se in ogni configurazione istantanea al più una mossa è possibile, *indeterministico* (o non deterministico) in caso contrario.

Un automa indeterministico è un modo astratto di rappresentare un algoritmo che in certe situazioni procede per tentativi, esaminando due o più strade alternative.

Una *configurazione* è *finale* se il controllo è in uno stato qualificato come finale e la testina d'ingresso è posta sul terminatore della stringa. In altre varianti, in alternativa al trovarsi in uno stato finale, la configurazione finale è caratterizzata da una particolare condizione imposta al contenuto della memoria: per esempio l'essere vuota o il contenere un simbolo specificato.

La *stringa sorgente* x è *accettata* se l'automa, partendo dalla configurazione iniziale con $\vdash x \dashv$ in ingresso, esegue un calcolo che lo porta in una configurazione finale (un automa indeterministico potrebbe raggiungere la configurazione finale in più modi diversi).

L'insieme delle stringhe accettate dall'automa costituisce il *linguaggio accettato* (riconosciuto, definito) da esso.

Il calcolo termina, o perché l'automa ha raggiunto una configurazione finale, o perché non può svolgere alcuna mossa, nel senso che la mossa non è definita in quella configurazione istantanea; nel qual caso la stringa d'ingresso non è accettata con tale calcolo.

Due automi che accettano lo stesso linguaggio sono detti *equivalenti*. Naturalmente non è detto che due automi equivalenti siano dello stesso tipo, né che abbiano la stessa complessità di calcolo.

Macchina di Turing

Lo schema precedente di automa riproduce sostanzialmente la macchina introdotta da A. Turing nel 1936 e oggi accettata come formalizzazione di ogni

procedura di calcolo sequenziale. La famiglia dei linguaggi accettati è detta *ricorsivamente enumerabile*.

Il linguaggio riconosciuto è detto *decidibile* se esiste una macchina di Turing che lo accetta e che si ferma per ogni stringa d'ingresso. La famiglia dei linguaggi decidibili è più ristretta di quella dei linguaggi ricorsivamente enumerabili.

Tale macchina ha interesse come riconoscitore di due delle famiglie di linguaggi della classificazione gerarchica di Chomsky (p. 87). Infatti i linguaggi generati dalle grammatiche del tipo 0 sono esattamente i linguaggi ricorsivamente enumerabili.

I linguaggi dipendenti dal contesto o del tipo 1 sono quelli riconosciuti da una macchina di Turing che usa un nastro di memoria di lunghezza proporzionale alla lunghezza della stringa da riconoscere.

La macchina di Turing non ha utilità pratica, ma è un riferimento mentale e un termine di confronto per gli automi più efficienti che si usano come riconoscitori dei linguaggi tecnici.

Riconsiderando la memoria ausiliaria, sono due i casi di interesse pratico per la compilazione. Se la memoria ausiliaria manca, l'automa generale diviene la ben nota macchina a stati finiti o automa finito, un modello fondamentale in ogni campo dell'informatica.

Se la memoria è organizzata a pila³ si ha l'automa a pila, importante per la compilazione in quanto è il riconoscitore dei linguaggi liberi.

Le famiglie di linguaggi che interessano la compilazione e i corrispondenti riconoscitori sono dunque dei casi molto speciali rispetto ai linguaggi ricorsivamente enumerabili. Dal punto di vista della teoria della complessità del calcolo, il problema del riconoscimento, che può essere intrattabile per i linguaggi ricorsivamente enumerabili, ha sempre complessità polinomiale per quelli oggetto di studio. In particolare, i linguaggi regolari, riconosciuti dagli automi finiti, sono una sottoclasse dei linguaggi riconoscibili in tempo reale da una macchina di Turing; mentre i linguaggi liberi sono una sottoclasse dei linguaggi di complessità temporale polinomiale su tale macchina.

3.3 Introduzione agli automi finiti

Gli automi finiti sono certamente un dispositivo fondamentale per il calcolo. Da un lato la loro teoria ha raggiunto una grande profondità matematica, dall'altro, le applicazioni sono numerosissime: il progetto digitale, i sistemi di controllo, i protocolli di comunicazione, lo studio dell'affidabilità dei sistemi, ecc. La presente esposizione si focalizza sui pochi aspetti che interessano direttamente il progettista dei linguaggi e dei compilatori, limitandosi a qualche richiamo delle proprietà teoriche essenziali.

Nell'ordine si espongono i diagrammi stato-transizione e gli automi finiti visti

³Push-down stack, LIFO o Last In First Out.

come riconoscitori di linguaggi definiti da grammatiche unilineari. Si esamina poi il comportamento deterministico e non, e il passaggio dal secondo al primo. Segue la costruzione del riconoscitore finito d'un linguaggio descritto da un'espressione regolare, e viceversa. Si approfondiscono poi le proprietà dei linguaggi regolari presentando la chiusura rispetto al complemento e alla intersezione.

Conformemente allo schema generale, un automa finito possiede: il nastro d'ingresso contenente la stringa sorgente $x \in \Sigma^*$; l'unità di controllo con la sua memoria limitata; la testina di lettura, inizialmente posta sul primo carattere di x , moventesi da sinistra a destra con una sola scansione, sino al termine di x (o al primo errore). Dopo la lettura d'un carattere l'automata aggiorna lo stato dell'unità di controllo. Alla fine della scansione, l'automata riconosce o meno x , a seconda dello stato in cui si trova.

Un *diagramma stato-transizione* è un grafo orientato rappresentante l'automata. I nodi del grafo sono gli stati dell'unità di controllo. Gli archi, etichettati con caratteri terminali, mostrano i cambiamenti di stato (*transizioni*) che avvengono alla lettura dei caratteri della stringa.

Esempio 3.1. Costanti numeriche decimali.

Il linguaggio L delle costanti numeriche decimali è così definito:

$\Sigma = \Delta \cup \{0, \bullet\}$, dove $\Delta = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ è una cifra diversa dallo zero;

$$L = (0 \cup \Delta(0 \cup \Delta)^*) \bullet (0 \cup \Delta)^+$$

Il suo riconoscitore è descritto dal *diagramma* o anche dalla *tabella stato-transizioni*:

diagramma stato-transizione

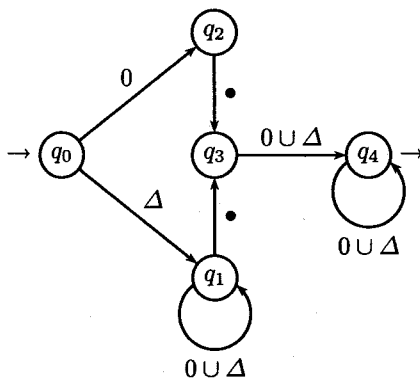


tabella stato-transizione

Stato Presente	Carattere corrente				
	0	1	...	9	•
→ q ₀	q ₂	q ₁	...	q ₁	—
q ₁	q ₁	q ₁	...	q ₁	q ₃
q ₂	—	—	...	—	q ₃
q ₃	q ₄	q ₄	...	q ₄	—
q ₄ →	q ₄	q ₄	...	q ₄	—

Per semplicità di disegno sono riuniti gli archi topologicamente eguali: ad es. l'arco ($q_0 \rightarrow q_1$) rappresenta un fascio di 9 archi, portanti le etichette $1, 2, \dots, 9$. Lo stato iniziale q_0 dell'automata è individuato dalla freccia entrante; la freccia uscente distingue lo stato finale (q_4), nel quale l'automata accetta la

stringa.

La matrice delle incidenze del grafo forma la tabella delle transizioni. Essa riporta per ogni coppia (stato presente, carattere corrente) il prossimo stato. Gli stati iniziali e finali sono distinti con le frecce.

La stringa sorgente $0 \bullet 2 \bullet$ fa transitare l'automa per gli stati q_0, q_2, q_3, q_4 , nell'ultimo dei quali non è però lecita la lettura del carattere \bullet . Poiché il calcolo si arresta prima di aver esaurito la scansione della stringa, essa non è riconosciuta. Invece la stringa $0 \bullet 21$ è accettata.

Se invece si volesse ridefinire il linguaggio per accettare anche le stringhe terminanti con \bullet , come $305\bullet$, basterebbe contrassegnare come finale anche lo stato q_3 .

Un automa può dunque avere più stati finali, ma un solo stato iniziale (altrimenti non sarebbe deterministico).

3.4 Automa finito deterministico

Si precisano ora i concetti introdotti nell'esempio precedente.

Un *automa M finito deterministico* è costituito da cinque entità:

1. Q , l'insieme degli stati (finito e non vuoto)
2. Σ , l'alfabeto di ingresso (o terminale)
3. la funzione di transizione $\delta : (Q \times \Sigma) \rightarrow Q$
4. $q_0 \in Q$, lo stato iniziale
5. $F \subseteq Q$, l'insieme degli stati finali.

La funzione δ specifica le mosse dell'automa: il significato di $\delta(q, a) = r$ è che M , trovandosi nello stato (presente) q e leggendo a , si porta nello stato (successivo) r . Se $\delta(q, a)$ non è definita, l'automa si ferma, e si può pensare che entri nello stato d'errore o rifiuto.

Per elaborare una stringa x non vuota, l'automa esegue una serie di mosse.

Sia ad es. $x = ab$; leggendo il primo carattere la mossa $\delta(q_0, a) = q_1$ porta nello stato q_1 , in cui si esegue la seconda mossa $\delta(q_1, b) = q_2$.

Per brevità, invece di scrivere $\delta(\delta(q_0, a), b) = q_2$, si combineranno le due mosse, scrivendo $\delta(q_0, ab) = q_2$, a indicare che la lettura della stringa ab porta nello stato q_2 : il secondo argomento della funzione delta può dunque essere una stringa.

Nel caso della stringa vuota si impone che la macchina resti sempre nello stato in cui si trova:

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

A seguito di queste estensioni, il dominio diviene $(Q \times \Sigma^*)$ e la funzione è definita da:

$$\forall \text{ carattere } a \in \Sigma, \forall \text{ stringa } y \in \Sigma^* :$$

$$\delta^*(q \bullet ya) = \delta(\delta^*(q, y), a)$$

Nella rappresentazione grafica dell'automa, se risulta $\delta(q, y) = q'$, allora, e solo allora, esiste un cammino dallo stato q allo stato q' , etichettato dalla stringa y , percorrendo il quale l'automa scandisce la stringa y . Quando l'automa percorre tale cammino, si dice che esegue un *calcolo*.

Una stringa x è *riconosciuta* (o accettata) dall'automa se scandendola l'automa cammina dallo stato iniziale a uno stato finale, $\delta(q_0, x) \in F$.

Come caso particolare, la stringa vuota è riconosciuta se, e solo se, lo stato iniziale è anche finale.

Il *linguaggio riconosciuto* dall'automa M è

$$L(M) = \{x \in \Sigma^* \mid x \text{ è riconosciuta da } M\}$$

I linguaggi riconosciuti da automi del tipo considerato sono detti *riconoscibili a stati finiti*.

► Due automi sono *equivalenti* se riconoscono lo stesso linguaggio.

Esempio 3.2. (Esempio 3.1 continuato).

L'automa M di p. 99 è definito da:

$$\begin{aligned} Q &= \{q_0, q_1, q_2, q_3, q_4\} \\ \Sigma &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, \bullet\} \\ q_0 &= q_0 \\ F &= \{q_4\} \end{aligned}$$

Esempi di transizioni:

$$\delta(q_0, 3 \bullet 1) = \delta(\delta(q_0, 3 \bullet), 1) = \delta(\delta(\delta(q_0, 3), \bullet), 1) = \delta(\delta(q_1, \bullet), 1) = \delta(q_3, 1) = q_4$$

Poiché $q_4 \in F$, la stringa $3 \bullet 1$ è accettata. Invece, poiché $\delta(q_0, 3 \bullet) = q_3$ non è finale, la stringa $3 \bullet$ non appartiene al linguaggio, né vi appartiene 02 in quanto $\delta(q_0, 02) = \delta(\delta(q_0, 0), 2) = \delta(q_2, 2)$, che non è definita.

L'efficienza di calcolo d'un automa deterministico è la migliore possibile in assoluto: infatti l'algoritmo decide l'appartenenza della stringa in tempo reale, scandendola una sola volta da sinistra a destra.

3.4.1 Stato di errore e completamento dell'automa

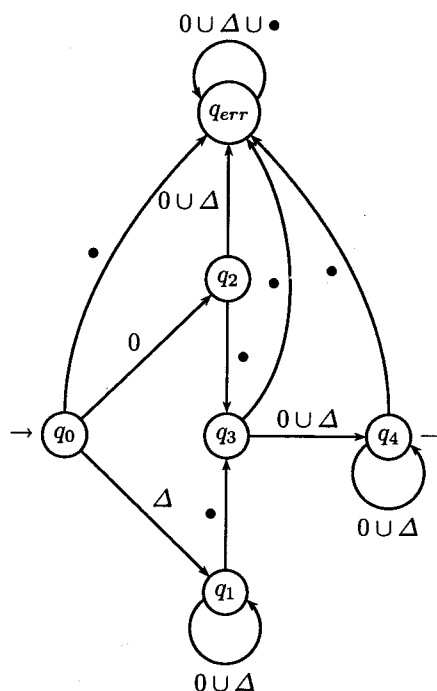
Se nello stato q la mossa non è definita per il carattere a , si può dire che l'automa alla lettura di tale carattere cade nello stato d'errore, q_{err} , dal quale non potrà più uscire, qualsiasi carattere sia successivamente letto. Per questa ragione tale stato è anche chiamato il *pozzo*.

► La funzione di transizione può essere sempre *completata* con lo stato di errore, senza modificare il linguaggio accettato, ponendo:

► \forall stato $q \in Q$ e \forall carattere $a \in \Sigma$ se $\delta(q, a)$ è indefinita, poni $\delta(q, a) = q_{err}$

$$\forall \text{ carattere } a \in \Sigma \text{ poni } \delta(q_{err}, a) = q_{err}$$

Il riconoscitore delle costanti decimali, nella versione completata con lo stato di errore, è:



Se un calcolo cade nello stato di errore non ne esce, né può raggiungere lo stato finale per riconoscere una stringa. Di conseguenza, l'automa completato accetta lo stesso linguaggio dell'automa originale. Convenzionalmente lo stato di errore è sempre sottinteso e non occorre disegnarlo.

3.4.2 Automa pulito

Un automa potrebbe contenere delle parti inutili, che non danno contributo al linguaggio riconosciuto e vanno normalmente eliminate. I concetti seguenti valgono anche per gli automi indeterministici e in generale per ogni tipo di automa.

Uno stato q è *raggiungibile* da uno stato p se esiste un calcolo che porta l'automa da p a q . Uno stato è *accessibile* se è raggiungibile dallo stato iniziale, è *post-accessibile* se da esso uno stato finale è raggiungibile. Uno stato accessibile e post-accessibile è detto *utile*. In altro modo, uno stato utile giace su un cammino tra lo stato iniziale e uno stato finale.

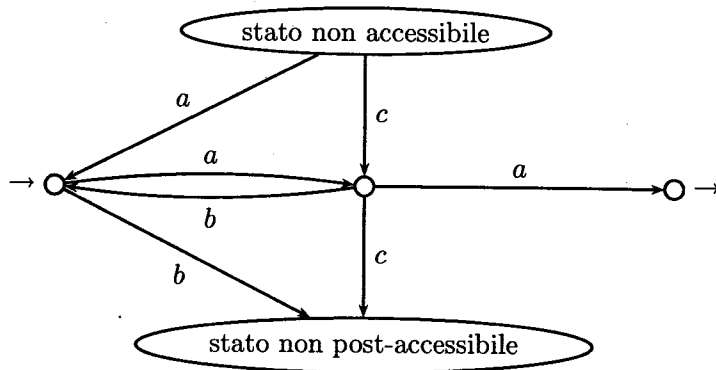
Un automa è *pulito* se ogni suo stato è utile.

Proprietà 3.3. Ogni automa finito ammette un automa pulito equivalente.

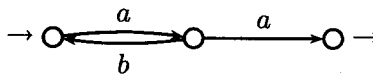
Per pulire l'automa, individuati gli stati non utili, li si sopprime con tutti gli archi incidenti, nel modo ora illustrato.

Esempio 3.4. Eliminazione degli stati inutili.

automa non pulito:



automa pulito:



3.4.3 Automa minimo

Ogni linguaggio può essere riconosciuto da tanti automi equivalenti pur se diversi rispetto al numero di stati o alla funzione di transizione. Ma l'automa più piccolo è unico, nel senso precisato dalla seguente affermazione.

Proprietà 3.5. Per ogni linguaggio a stati finiti, il riconoscitore deterministico minimo rispetto al numero degli stati è unico (a meno d'una ridenominazione degli stati).

Prima di descrivere un algoritmo di minimizzazione⁴ di un automa, è necessaria una nuova definizione.

Si suppone che l'automa sia pulito. Tuttavia esso può contenere due stati che, ai fini del riconoscimento, possono essere fusi in un solo stato. Come individuarli? Si definisce una relazione binaria, detta di *indistinguibilità* o di Nerode, tra coppie di stati.

Definizione 3.6. Gli stati p e q sono *indistinguibili* se, e solo se, per ogni stringa $x \in \Sigma^*$, o entrambi gli stati $\delta(p, x)$ e $\delta(q, x)$ sono finali, o nessuno dei due lo è.

La relazione complementare è chiamata *distinguibilità*.

Parafrasando la definizione, due stati p, q sono *indistinguibili* se, partendo rispettivamente dall'uno e dall'altro e scandendo la stessa stringa x non è possibile giungere a due stati, uno finale, l'altro non.

Si osserva che:

⁴Il problema della minimizzazione è stato risolto anche con altri algoritmi. Si veda la rassegna in [52].

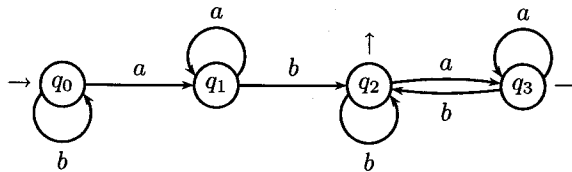
1. lo stato pozzo q_{err} è distinguibile da ogni altro stato p , poiché certamente esiste una stringa x per cui risulta $\delta(p, x) \in F$, mentre per ogni stringa x , è $\delta(q_{err}, x) = q_{err}$;
2. p e q sono distinguibili se p è finale e q no, poiché $\delta(p, \epsilon) \in F$ e $\delta(q, \epsilon) \notin F$;
3. p e q sono distinguibili se, per qualche carattere a , gli stati prossimi $\delta(p, a)$ e $\delta(q, a)$ sono distinguibili.

In particolare p è distinguibile da q se gli insiemi delle etichette delle frecce uscenti rispettivamente da p e da q sono diversi. Infatti in tale caso, esiste un carattere a , tale che la mossa dallo stato p conduce nello stato p' , mentre la mossa da q non è definita (ossia conduce nello stato di errore); per la condizione 3. tali stati sono distinguibili.

L'indistinguibilità è una relazione transitiva e riflessiva, le cui classi di equivalenza possono essere calcolate con un semplice procedimento che si descrive con un esempio.

Esempio 3.7. Classi di equivalenza degli stati indistinguibili.

Si consideri l'automa M :



Si costruisce ora la tabella di dimensioni $|Q| \times |Q|$ della relazione di indistinguibilità. È inutile riempire le caselle poste sopra la diagonale principale, essendo la relazione simmetrica.

Si marcherà la casella (p, q) con una X se la coppia di stati è distinguibile. Si inizializzano con X le caselle per le quali uno solo degli stati è finale:

q_1		-	-	-
q_2	X	X	-	-
q_3	X	X		-
	q_0	q_1	q_2	q_3

Quindi si esamina ogni casella (p, q) non marcata e, per ogni carattere terminale a , si rilevano gli stati successivi $r = \delta(p, a)$ e $s = \delta(q, a)$.

Se la casella (r, s) è già marcata X , cioè r è distinguibile da s , allora lo sono anche p e q , e la casella (p, q) è da marcare con X .

Se invece la casella (r, s) non è marcata con X , si inserisce nella casella (p, q) la coppia (r, s) , come promemoria che in futuro, se si marcherà (r, s) , si dovrà marcare anche (p, q) .

Le coppie di stati prossimi sono indicate nella tabella di sinistra. Il risultato

del passo è la tabella di destra, che ha la marca nella casella (1,0) poiché la coppia $(0, 2) \equiv (2, 0)$ era già marcata:

q_1	$(1,1),(0,2)$	—	—	—
q_2	X	X	—	—
q_3	X	X	$(3,3),(2,2)$	—
	q_0	q_1	q_2	q_3

q_1	X	—	—	—
q_2	X	X	—	—
q_3	X	X	$(3,3),(2,2)$	—
	q_0	q_1	q_2	q_3

Tutte le caselle sono marcate e l'algoritmo termina.

Le caselle non marcate con X designano le coppie indistinguibili, nell'es. la coppia (q_2, q_3) .

Una classe d'equivalenza della relazione di indistinguibilità contiene tutti gli stati tra loro indistinguibili.

Nell'esempio le classi d'equivalenza sono $[q_0]$, $[q_1]$, $[q_2, q_3]$.

È interessante esaminare anche un caso in cui la funzione di transizione non è totale. Si modifichi l'automa M cancellando l'autoanello $\delta(q_3, a) = q_3$, ossia ridefinendo la funzione come $\delta(q_3, a) = q_{err}$. Di conseguenza gli stati q_2, q_3 diventano distinguibili, poiché $\delta(q_2, a) = q_3$, $\delta(q_3, a) = q_{err}$. Le classi di equivalenza sono tutte unitarie.

Costruzione dell'automa minimo

L'automa minimo M' equivalente a quello dato M ha come stati le classi di equivalenza della relazione di indistinguibilità. Per definire la funzione di transizione basta dire che vi è in M' un arco tra due classi di equivalenza

$$[\dots, p_r, \dots] \xrightarrow{b} [\dots, q_s, \dots]$$

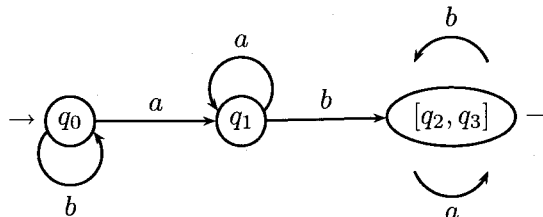
se, e solo se, in M vi è un arco

$$p_r \xrightarrow{b} q_s$$

tra due stati, ordinatamente appartenenti alle due classi di equivalenza; si noti che lo stesso arco di M' deriva in generale da più archi di M .

Esempio 3.8. (Esempio 3.7 continuato).

L'automa minimo M' è:



Questo è l'automa più piccolo equivalente a quello dato. Infatti sarebbe facile constatare che, fondendo insieme due stati non equivalenti rispetto all'indistinguibilità, si ottiene una macchina che riconosce un linguaggio più ampio dell'originale.

L'esistenza dell'algoritmo di minimizzazione dimostra la proprietà 3.5 d'unicità dell'automa minimo equivalente a un automa dato, sulla quale si basa un test per verificare se due automi sono equivalenti: si minimizzano quindi si confrontano, per vedere se sono identici, a meno d'un cambiamento di nome degli stati.⁵

Nelle applicazioni, è ovvio che ragioni di economia fanno preferire l'automa minimo, ma il risparmio può essere trascurabile, soprattutto per i casi che interessano la compilazione.

Addirittura in certe situazioni la minimizzazione degli stati del riconoscitore è da evitare: se l'automa è arricchito con certe azioni per il calcolo d'una funzione di uscita (funzione semantica), come si vedrà nel capitolo 5, due stati, che sarebbero indistinguibili per il riconoscitore, potrebbero essere associati a azioni diverse, con la conseguenza che, fondendoli insieme, si confonderebbero le azioni da compiere.

Infine si anticipa che la proprietà d'unicità dell'automa minimo non vale in generale per gli automi non deterministici di prossima introduzione.

3.4.4 Dall'automa alla grammatica

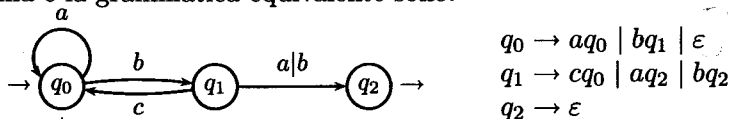
Sarà facile constatare che gli automi a stati finiti e le grammatiche unilineari (o del tipo 3 di Chomsky) di p. 70 definiscono esattamente la stessa famiglia di linguaggi.

Per primo si mostra come costruire una grammatica lineare a destra equivalente a un automa. La grammatica G ha come simboli nonterminali gli stati Q dell'automa, come assioma lo stato iniziale, per ogni mossa $q \xrightarrow{a} r$ ha la regola $q \rightarrow ar$, e, se lo stato q è finale, ha anche la regola terminale $q \rightarrow \varepsilon$.

È immediato vedere che vi è corrispondenza biunivoca tra i calcoli dell'automa e le derivazioni della grammatica: una stringa x è accettata dall'automa se, e solo se, esiste la derivazione $q_0 \xRightarrow{+} x$.

Esempio 3.9. Dall'automa alla grammatica lineare a destra.

L'automa e la grammatica equivalente sono:



La frase bca , riconosciuta in 3 passi dall'automa, deriva dall'assioma in $3 + 1$ passi:

⁵Esistono tuttavia algoritmi più efficienti per decidere l'equivalenza di due automi senza prima minimizzarli.

$$q_0 \Rightarrow bq_1 \Rightarrow bcq_0 \Rightarrow bcaq_0 \Rightarrow bcae$$

Si osservi che la grammatica possiede regole vuote, ma essa può essere convertita nella forma non annullabile. Un modo è di applicare la trasformazione di p. 60.

Nell'es. l'insieme dei nonterminali annullabili è $Null = \{q_0, q_2\}$, da cui si costruiscono le regole equivalenti:

$$q_0 \rightarrow aq_0 \mid bq_1 \mid a \mid \varepsilon \quad q_1 \rightarrow cq_0 \mid aq_2 \mid bq_2 \mid a \mid b \mid c$$

Ma q_2 , avendo solo la regola vuota, è da eliminare, e la conseguente ripulitura produce la grammatica

$$q_0 \rightarrow aq_0 \mid bq_1 \mid a \mid \varepsilon \quad q_1 \rightarrow cq_0 \mid a \mid b \mid c$$

Si noti che la regola nulla $q_0 \rightarrow \varepsilon$ non può essere tolta perché q_0 è l'assioma, ossia la stringa vuota sta nel linguaggio.

In questa versione della grammatica è facile vedere che, a una mossa entrante in uno stato finale $(q) \xrightarrow{a} (r)$, possono corrispondere le due regole

$q \rightarrow ar \mid a$, una terminale, l'altra di transizione verso lo stato r .

Se il passaggio dall'automa alla grammatica equivalente è stato immediato, quello inverso, dalla grammatica all'automa, richiederà la modifica della definizione della macchina per incorporarvi il concetto di indeterminismo.

3.5 Automi indeterministici

In una grammatica lineare a destra possono trovarsi due alternative

$$A \rightarrow aB \mid aC \quad \text{dove } a \in \Sigma, A, B, C \in V \quad \begin{array}{c} (C) \xleftarrow{a} (A) \xrightarrow{a} (B) \end{array}$$

inizianti con lo stesso carattere a . La precedente corrispondenza, tra regole e mosse dell'automa, porta a tracciare due frecce con la stessa etichetta, dallo stato A a due stati diversi B e C .

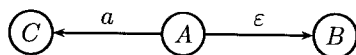
Nello stato A , leggendo il carattere a , la macchina può scegliere in quale degli stati andare, e il suo comportamento non è deterministico. Formalmente la funzione di transizione prende due valori $\delta(A, a) = \{B, C\}$.

Similmente una regola grammaticale di copiatura

$$A \rightarrow B \quad \text{dove } B \in V \quad \begin{array}{c} (A) \xrightarrow{\varepsilon} (B) \end{array}$$

porta a disegnare la transizione dallo stato A allo stato B , senza lettura d'un carattere, o, che è lo stesso, con la lettura della stringa vuota.

Una mossa che avviene senza leggere un carattere d'ingresso è detta *spontanea* o *mossa epsilon*. Anche le mosse spontanee possono rendere indeterministico il funzionamento, come nel caso



dove nello stato A l'automata può scegliere di andare spontaneamente (cioè senza lettura) in B , oppure di leggere un carattere a , e se esso è a , di andare in C .

3.5.1 Motivazioni dell'indeterminismo

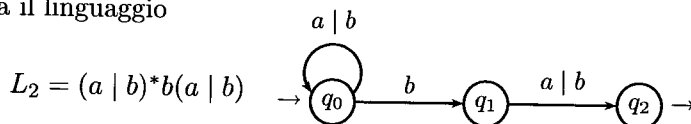
Si è appena visto che la corrispondenza tra grammatiche e automi introduce in modo naturale le transizioni con più stati d'arrivo e le mosse spontanee, le due principali situazioni indeterministiche. Ma, poiché il concetto di indeterminismo potrebbe apparire lontano dalla pratica, è opportuno motivarlo ulteriormente.

Concisione

La definizione di un linguaggio mediante una macchina non deterministica, può risultare più leggibile e compatta, come mostra il prossimo esempio.

Esempio 3.10. Penultimo carattere.

Sono da definire le stringhe, come $abaabb$, aventi una b nella penultima posizione, ossia il linguaggio



L'automata indeterministico N_2 opera nel seguente modo: data una stringa, cerca di trovare un cammino, dallo stato iniziale a quello finale, etichettato con la stringa data; in caso di successo, la stringa è accettata. Così la stringa $baba$ è riconosciuta dal calcolo

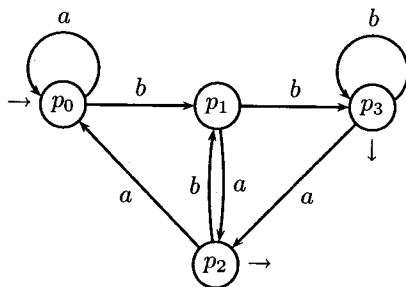
$$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2$$

pur se altri calcoli possibili, come

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

falliscono perché non conducono allo stato finale.

Lo stesso linguaggio è accettato dall'automata deterministico M_2 :



Non solo questa macchina ha un numero maggiore di stati, ma essa non rende altrettanto evidente la condizione che il penultimo carattere sia b .

Generalizzando l'esempio, dal linguaggio L_2 al linguaggio L_k tale che il k -ultimo, $k \geq 2$, carattere sia b , si vede subito che l'automa indeterministico ha $k + 1$ stati, mentre si potrebbe dimostrare che il numero di stati dell'automa deterministico minimo è dato da una funzione che cresce esponenzialmente con k : una dimostrazione che l'indeterminismo può rendere molto più concise le definizioni.

Dualità sinistra - destra

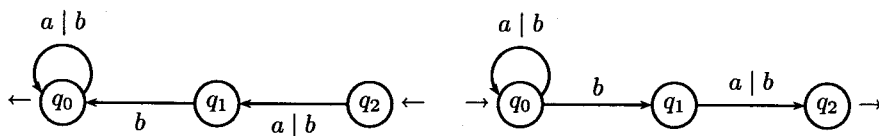
Un'altra situazione che conduce all'indeterminismo nasce nel passaggio dal riconoscitore deterministico d'un linguaggio L al riconoscitore del linguaggio speculare L^R , quello che scandisce il testo dalla fine all'inizio (come talvolta è richiesto). Tale riconoscitore si ottiene facilmente nel seguente modo: si scambiano gli stati iniziali e finali ⁶ e si inverte il senso delle frecce. Entrambe le operazioni possono causare situazioni indeterminate

Esempio 3.11. Il linguaggio avente b come penultimo carattere (L_2 dell'es. precedente) è l'immagine riflessa del linguaggio in cui il secondo carattere è b

$$L' = \{x \in \{a, b\}^* \mid b \text{ è il secondo carattere di } x\} \quad L_2 = (L')^R$$

L' è riconosciuto dall'evidente automa deterministico (a sinistra):

⁶Se l'automa ha più stati finali, gli stati iniziali sono multipli, come si vedrà.



Applicando all'automa deterministico la trasformazione speculare, si ottiene l'automa indeterministico (a destra), che coincide con quello di p. 108.

Come ultima motivazione, si anticipa che il passaggio attraverso gli automi indeterministici è utilizzato nella costruzione del riconoscitore del linguaggio definito da un'espressione regolare.

3.5.2 Riconoscimento indeterministico

Si precisano qui i concetti del calcolo non deterministico a stati finiti, ignorando per ora le mosse spontanee.

Un *automa indeterministico* N a stati finiti, senza mosse spontanee, è definito da

- l'insieme degli stati Q
- l'alfabeto terminale Σ
- due sottoinsiemi di Q : l'insieme I degli stati *iniziali* e l'insieme F di quelli *finali*
- la relazione di transizione δ , sottoinsieme del prodotto cartesiano $Q \times \Sigma \times Q$.

Note. Questa macchina può avere più d'uno stato iniziale. Nella rappresentazione grafica una transizione (q_1, a, q_2) è disegnata come un arco etichettato con a , dal primo nodo al secondo.

Al solito, un *calcolo* della macchina è una serie di transizioni tali che l'origine di ciascuna coincide con la destinazione della precedente

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

L'*origine* del calcolo è q_0 , il *termine* è q_n e la *lunghezza* è il numero di transizioni n .

I calcoli di lunghezza 1 sono le transizioni. L'*etichetta* del calcolo è il concatenamento $a_1 a_2 \dots a_n$ dei caratteri letti a ogni transizione.

Il calcolo, in forma abbreviata, è denotato dalla scrittura $q_0 \xrightarrow{a_1 a_2 \dots a_n} q_n$.

Una stringa x è *riconosciuta* (o accettata) dall'automa se essa è l'etichetta d'un calcolo che ha origine in uno stato iniziale, termina in uno stato finale e ha x come etichetta.

Convenzionalmente, ogni stato è origine e termine d'un calcolo di lunghezza 0, avente come etichetta la stringa vuota ε . Di conseguenza la stringa vuota

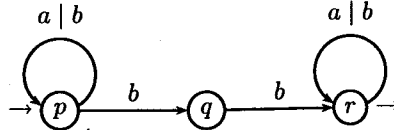
è accettata dall'automata se, e soltanto se, uno stato iniziale è anche finale.

Il linguaggio $L(N)$ riconosciuto dall'automata N è l'insieme delle stringhe riconosciute

$$L(N) = \{x \in \Sigma^* \mid q \xrightarrow{x} r \text{ con } q \in I, r \in F\}$$

Esempio 3.12. Ricerca d'una parola in un testo.

Data una parola y , per riconoscere se un testo la contiene, basta sottoporlo all'automata che accetta il linguaggio $(a \mid b)^*y(a \mid b)^*$. Si presenta per la parola $y = bb$ l'automata:



La stringa $abbb$ è l'etichetta di più calcoli originanti nello stato iniziale:

$$\begin{array}{ll} p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{b} p \xrightarrow{b} p & p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{b} p \xrightarrow{b} q \\ p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{b} q \xrightarrow{b} r & p \xrightarrow{a} p \xrightarrow{b} q \xrightarrow{b} r \xrightarrow{b} r \end{array}$$

I primi due calcoli non trovano la parola cercata. Gli ultimi due la trovano rispettivamente nelle posizioni $ab \underbrace{bb}$ e $a \underbrace{bb} b$.

Funzione di transizione

Le mosse dell'automata possono essere definite tramite una funzione di transizione, pur di ammettere funzioni a più valori.

Per un automa indeterministico $N = (Q, \Sigma, \delta, I, F)$, privo di mosse spontanee, la funzione δ è definita nel dominio e codominio:

$$\delta : (Q \times (\Sigma \cup \epsilon)) \rightarrow \text{parti finite di } Q$$

Per un carattere terminale a , il significato di

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

è che la macchina, dallo stato presente q , leggendo a , può arbitrariamente andare in uno dei k stati p_1, \dots, p_k .

Si definisce poi la funzione anche per una stringa y qualsiasi:

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

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

ovvero $p \in \delta(q, y)$ se vi è un calcolo etichettato y , da q a p .

Mediante la funzione di transizione, si scrive la seguente definizione del linguaggio accettato dall'automata N :

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

cioè l'insieme calcolato dalla funzione di transizione deve contenere uno stato finale, affinché la stringa x sia accettata.

Esempio 3.13. (Esempio 3.12 continuato.)

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

3.5.3 Automi con mosse spontanee

Un altro comportamento indeterministico è quello d'un automa che cambia stato, pur senza leggere un carattere, compiendo così una mossa spontanea. Essa nel diagramma stato-transizione è rappresentata da un arco etichettato ε (o ε -arco).

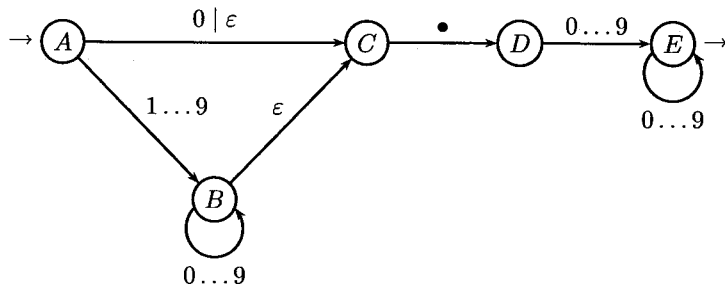
Con l'uso di ε -archi è facile costruire i riconoscitori di linguaggi ottenuti per composizione regolare di altri linguaggi. Il prossimo esempio mostra il caso dell'unione e del concatenamento.

Esempio 3.14. Costanti decimali.

Il linguaggio comprende le costanti quali ad es. $90 \bullet 01$. La parte intera a sinistra del punto decimale può essere 0 o scomparire (es. $\bullet 01$); ma non può contenere zeri non significativi. Il linguaggio è prodotto dalla e.r.

$$L = (0 \mid \varepsilon \mid N) \bullet (0 \dots 9)^+ \text{ dove } N = (1 \dots 9)(0 \dots 9)^*$$

Il seguente automa riproduce direttamente la struttura della definizione.



Si noti che, scegliendo la mossa spontanea da A a C, scompare la parte intera; l'altra mossa spontanea, da B a C, impone che la parte intera N sia seguita dal punto decimale. La stringa $34 \bullet 5$ è accettata dal calcolo

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

La presenza di mosse spontanee non modifica il concetto di riconoscimento: una stringa x è *riconosciuta* dall'automato con mosse spontanee se essa è l'etichetta d'un calcolo che ha origine in uno stato iniziale, termina in uno stato finale e ha x come etichetta.

Ora però la lunghezza del calcolo (ossia il numero degli archi del cammino percorso nel grafo) può superare quella della stringa, a causa della presenza

di ϵ -archi. Di conseguenza il numero di passi dell'algoritmo può superare la lunghezza della stringa sorgente, e di conseguenza l'automa non funziona più in tempo reale. Esso però ha sempre una complessità di calcolo lineare, perché si può escludere che vengano eseguiti dei cicli di mosse spontanee.

Unicità dello stato iniziale

Nelle definizioni l'automa indeterministico può avere più stati iniziali, ma è facile ottenere un automa equivalente con stato iniziale unico. Basta aggiungere il nuovo stato q_0 iniziale, che sarà l'unico, e le ϵ -mosse che da esso vanno agli stati ex iniziali dell'automa da trasformare. Chiaramente un calcolo del nuovo automa accetta una stringa se, e soltanto se, anche il vecchio automa la accetta.

La mosse spontanee così aggiunte potranno poi essere eliminate, come si vedrà a p. 119.

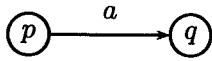
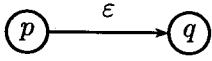

3.5.4 Corrispondenza tra automa e grammatica

Si riepilogano le corrispondenze tra automi indeterministici, anche con mosse spontanee, e grammatiche unilineari, confermando che la differenza tra i due modelli è una mera variante descrittiva.

Confrontando dunque una grammatica e un automa, si esplicitano le condizioni che rendono equivalenti le loro regole e mosse rispettive.

Denotiamo con $G = (V, \Sigma, P, S)$ la grammatica lineare a destra e con $N = (Q, \Sigma, \delta, q_0, F)$ l'automa. Possiamo supporre, per quanto detto sopra, che lo stato iniziale sia unico.

Si suppone inizialmente che le regole della grammatica siano strettamente unilineari (p. 71). Ecco la corrispondenza:

	<i>Grammatica lineare a destra</i>	<i>Automa finito</i>
1	Alfabeto nonterminale V	Insieme degli stati $Q = V$
2	Assioma $S = q_0$	Stato iniziale $q_0 = S$
3	$p \rightarrow aq$, dove $a \in \Sigma$ e $p, q \in V$	
4	$p \rightarrow q$, dove $p, q \in V$	
5	$p \rightarrow \epsilon$	Stato finale 

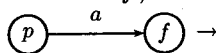
Gli stati Q dell'automa sono in corrispondenza biunivoca con i simboli non-terminali V della grammatica. Lo stato iniziale corrisponde all'assioma.

Si noti (riga 3) che una coppia di alternative $p \rightarrow aq \mid ar$ corrisponde a due mosse non deterministiche. Una regola di copiatura (riga 4) corrisponde a una mossa spontanea. Uno stato è finale (riga 5) se corrisponde a un nonterminale avente una regola vuota.

È facile constatare che ogni derivazione della grammatica corrisponde a un calcolo dell'automa, e viceversa; di conseguenza i due modelli definiscono lo stesso linguaggio.

Proprietà 3.15. Un linguaggio è riconosciuto da un automa finito se, e solo se, è generato da una grammatica unilineare.

Se la grammatica possiede anche delle regole terminali del tipo $p \rightarrow a$, dove $a \in \Sigma$, l'automa conterrà anche uno stato finale f , distinto da quelli prodotti dalla riga 5 della tabella, e la mossa



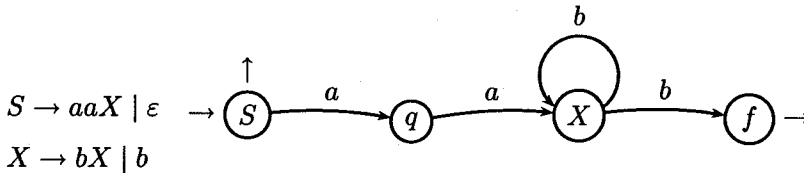
Esempio 3.16. Equivalenza tra grammatiche lineari a destra e automi.

La grammatica corrispondente all'automa delle costanti decimali di p. 112 è:

$$\begin{array}{ll}
 A \rightarrow 0C \mid C \mid 1B \mid \dots \mid 9B & B \rightarrow 0B \mid \dots \mid 9B \mid C \\
 C \rightarrow \bullet D & D \rightarrow 0E \mid \dots \mid 9E \\
 E \rightarrow 0E \mid \dots \mid 9E \mid \epsilon
 \end{array}$$

dove A è l'assioma.

Come secondo esempio, nella grammatica lineare a destra



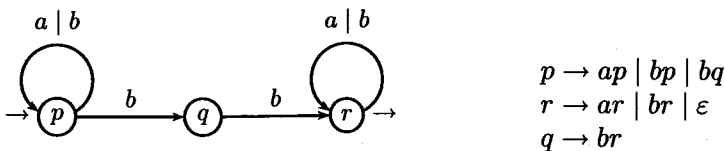
anche se la regola $S \rightarrow aaX$ non è strettamente lineare, un semplice accorgimento ne permette la traduzione (mostrata a destra) in una serie di due archi: basta creare uno stato intermedio q dopo la prima a . Inoltre si crea il nuovo stato finale f per riprodurre l'effetto della regola $X \rightarrow b$.

3.5.5 Ambiguità dell'automa

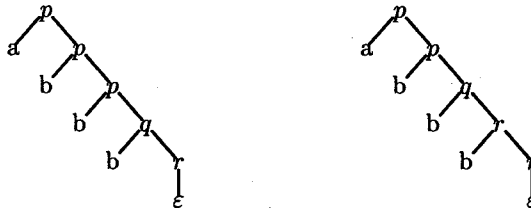
Un *automa* è detto *ambiguo* se accetta una frase con due calcoli diversi. Grazie alla corrispondenza biunivoca tra i calcoli e le derivazioni della grammatica, si può dire che un automa è ambiguo se, e soltanto se, la grammatica lineare a destra corrispondente è essa stessa ambigua, ossia se genera una frase con due alberi sintattici diversi.

Esempio 3.17. Ambiguità dell'automa e della grammatica.

Nell'esempio 3.12 di p. 111 si è visto che l'automa



riconosce la frase $abbb$ con due calcoli. La corrispondente grammatica (a destra) genera la stessa frase con i due alberi:



3.5.6 Grammatica lineare a sinistra e automa

Poiché si sa che la famiglia *REG* è pure definita dalle grammatiche lineari a sinistra, si possono enunciare le regole di corrispondenza tra le grammatiche di tale tipo e gli automi finiti, seguendo un principio di dualità tra sinistra e destra.

Ora le regole della grammatica G hanno le forme:

$$A \rightarrow Ba, \quad A \rightarrow B, \quad A \rightarrow \varepsilon$$

Si consideri il linguaggio speculare $L^R = (L(G))^R$: esso è generato dalla grammatica riflessa, denotata G_R , ottenuta (p. 79) cambiando le regole della prima forma in $A \rightarrow aB$, mentre le altre due forme non mutano. Poiché evidentemente la grammatica riflessa G_R è lineare a destra, si sa costruire il riconoscitore N_R di L^R . Non resta che trasformarlo, invertendo le frecce e scambiando stati finali e iniziali, per ottenere il riconoscitore del linguaggio originale L .

Esempio 3.18. Dalla grammatica lineare a sinistra all'automa.
Data la grammatica G :

$$S \rightarrow Aa \mid Ab \quad A \rightarrow Bb \quad B \rightarrow Ba \mid Bb \mid \varepsilon$$

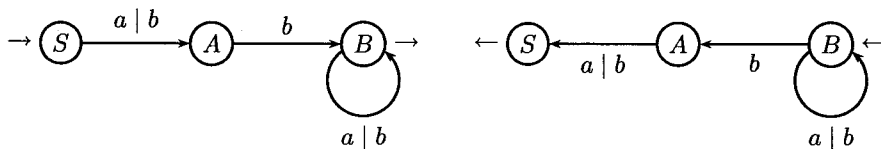
invertendo specularmente le regole si ottiene la grammatica riflessa G_R :

$$S \rightarrow aA \mid bA \quad A \rightarrow bB \quad B \rightarrow aB \mid bB \mid \varepsilon$$

dalla quale si costruisce l'automa N^R e, con l'inversione delle frecce e lo scambio tra stati iniziali e finali, il riconoscitore N del linguaggio $L(G)$:

Riconoscitore N_R di $(L(G))^R$

Riconoscitore N di $L(G)$



Per inciso, il linguaggio è quello in cui il penultimo carattere è b .

3.6 Dall'automa all'espressione regolare direttamente: il metodo BMC

Poiché un automa finito equivale a una grammatica lineare a destra, e questa genera un linguaggio regolare, l'e.r. del linguaggio riconosciuto dall'automa può essere ottenuta risolvendo le equazioni lineari con il procedimento di p. 71, ma si offrirà qui anche un percorso diretto, il metodo di eliminazione di Brzozowski e McCluskey (BMC).

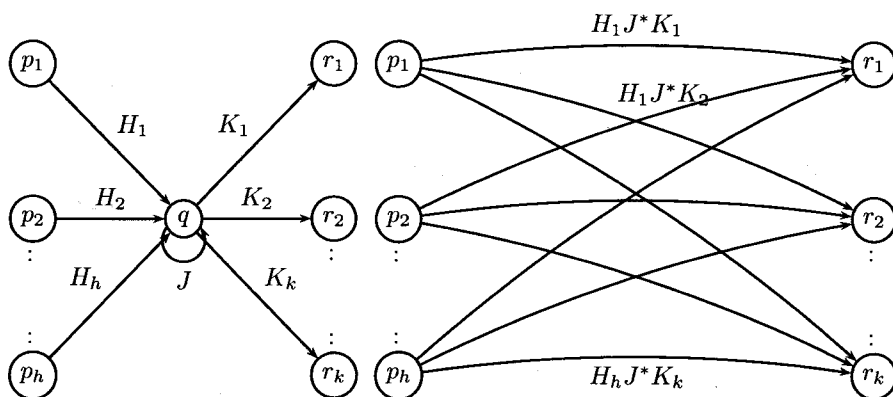
Si suppone che lo stato iniziale i sia unico e privo di archi entranti, e lo stato finale t sia unico e privo di archi uscenti. Se così non fosse, basterebbe creare il nuovo stato i e collegarlo con mosse spontanee agli stati ex-iniziali dell'automa; e similmente per il nuovo stato finale t .

Gli stati diversi da i e da t sono detti interni.

Si costruirà un automa equivalente a quello dato, in cui però gli archi possono essere etichettati non soltanto con caratteri terminali, ma anche con linguaggi regolari. Esso è chiamato *automa generalizzato*.

L'idea è di eliminare uno alla volta gli stati interni, aggiungendo delle mosse compensatorie, che preservano il linguaggio riconosciuto, etichettate con e.r.; ciò fino a quando restano soltanto gli stati iniziale e finale. A quel punto l'etichetta dell'arco $i \rightarrow t$ è la e.r. del linguaggio.

Si veda, a sinistra, uno stato interno q con gli archi incidenti; a destra, lo stesso frammento di automa, dopo l'eliminazione dello stato q e l'aggiunta di archi etichettati con le stesse stringhe prodotte dall'attraversamento di q :



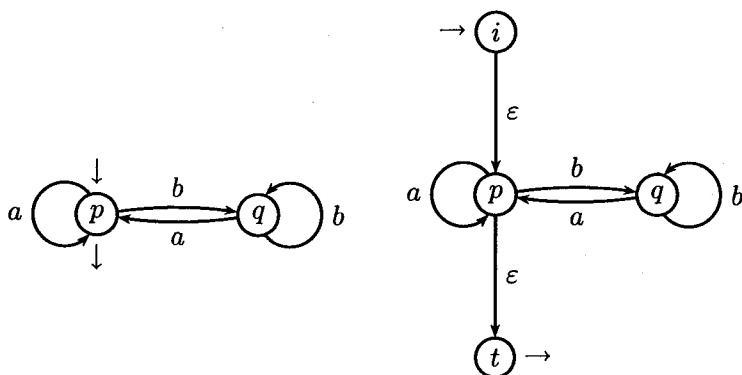
Per non complicare la figura, alcune etichette sono omesse, chiarendo che, per ogni coppia di stati p_i, r_j , vi è l'arco $p_i \xrightarrow{H_i J^* K_j} r_j$.

Si noti che alcuni stati p_i, r_j potrebbero essere coincidenti.

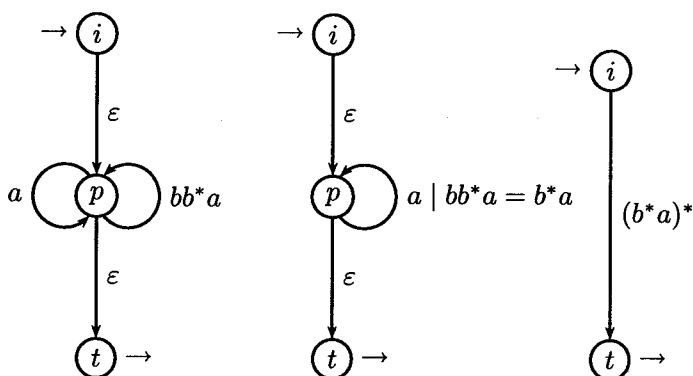
È evidente che l'insieme delle stringhe, che possono essere lette dall'automata originale passando da uno stato p_i a uno stato r_j , coincide con il linguaggio che etichetta l'arco $p_i \rightarrow r_j$ dell'automata trasformato.

Al fine di calcolare la e.r. del linguaggio riconosciuto da un automa, si applica la trasformazione tante volte, quanti sono gli stati interni, ottenendo al termine un automa con i soli stati iniziale e finale, un solo arco e la e.r. del linguaggio come sua etichetta.

Esempio 3.19. (Sakarovitch). L'automata dato è mostrato prima e dopo la normalizzazione:



Si applica l'algoritmo BMC nell'ordine q, p :



L'ordine di eliminazione degli stati interni è irrilevante, ma, come già nella risoluzione dei sistemi di equazioni lineari, ordini diversi possono produrre e.r. di diversa complessità, pur se equivalenti. Così nell'esempio precedente l'ordine p, q produrrebbe la e.r. $(a^*b)^+a^+ \mid a^*$.

3.7 Eliminazione dell'indeterminismo

Pur se in certe situazioni la formulazione più diretta del linguaggio desiderato è quella non deterministica, la versione finale del riconoscitore del linguaggio deve quasi sempre essere deterministica, per ragioni di efficienza. Fanno eccezione quei casi in cui il costo da minimizzare è il tempo di costruzione dell'automa invece del tempo di riconoscimento delle frasi, come ad es. quando l'automa viene usato in videoscrittura per ricercare nel testo una sola volta una parola specificata dall'utente.

Si mostrerà ora, attraverso un procedimento costruttivo, che ogni automa indeterministico può essere trasformato in uno deterministico equivalente, e, come corollario, che ogni grammatica unilineare ammette una grammatica equivalente non ambigua.

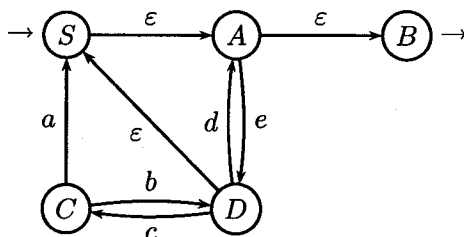
La trasformazione di un automa indeterministico in uno deterministico procede in due fasi:

1. Eliminazione delle mosse spontanee, ottenendo un automa in generale indeterministico: poiché le mosse spontanee corrispondono alle regole di copiatura della grammatica, si può applicare la trasformazione grammaticale che elimina queste ultime (p. 61).
2. Sostituzione di più transizioni non deterministiche con una sola che porta in un nuovo stato: questa fase è detta costruzione delle parti finite, perché i nuovi stati introdotti sono in corrispondenza con i sottoinsiemi dell'insieme degli stati.

Eliminazione delle mosse spontanee

Per la prima fase si passa subito all'esemplificazione, essendo l'applicazione della trasformazione grammaticale che elimina le regole di copiatura.

Esempio 3.20. Dato l'automa



si devono eliminare le regole di copiatura⁷ dalla grammatica equivalente:

$$\begin{array}{lll}
 S \rightarrow A & A \rightarrow B \mid eD & B \rightarrow \epsilon \\
 C \rightarrow aS \mid bD & D \rightarrow S \mid cC \mid dA &
 \end{array}$$

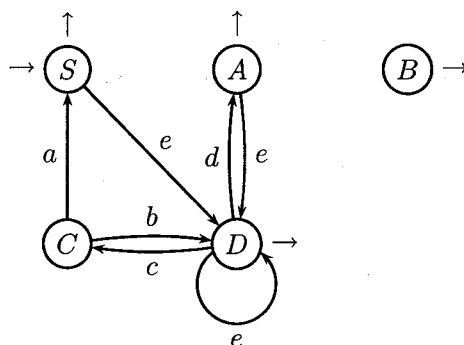
Calcolati gli insiemi delle copie

	<i>copia</i>
<i>S</i>	<i>S, A, B</i>
<i>A</i>	<i>A, B</i>
<i>B</i>	<i>B</i>
<i>C</i>	<i>C</i>
<i>D</i>	<i>D, S, A, B</i>

si ottiene la grammatica e il corrispondente automa senza mosse spontanee:

⁷Pur in presenza di ϵ -regole, l'algoritmo di p. 61 resta valido, poiché in una grammatica lineare ogni parte destra contiene non più d'un nonterminale.

$$\begin{array}{lll}
 S \rightarrow \varepsilon \mid eD & A \rightarrow \varepsilon \mid eD & B \rightarrow \varepsilon \\
 C \rightarrow aS \mid bD & D \rightarrow \varepsilon \mid eD \mid cC \mid dA &
 \end{array}$$



dove lo stato B , irraggiungibile dallo stato iniziale, è da eliminare.

Si noti che nel grafo dell'automa un intero cammino (ad es. $D \xrightarrow{\varepsilon} S \xrightarrow{\varepsilon} A \xrightarrow{\varepsilon} D$), composto di una catena di mosse spontanee concluse da una mossa di lettura, è sostituito da una mossa diretta ($D \xrightarrow{\varepsilon} D$). Se, dopo l'eliminazione delle mosse spontanee, l'automa risulta indeterministico, si applica la fase seguente.

3.7.1 Costruzione delle parti finite raggiungibili

Dato N , un automa indeterministico privo di mosse spontanee, si costruisce l'automa deterministico equivalente M' .

La costruzione si basa su quest'idea: se in N vi sono le mosse

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

poiché, dopo la lettura di a , non si sa in quale degli stati successivi p_1, p_2, \dots, p_k la macchina N si trovi, si crea in M' un nuovo stato collettivo, denominato

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

per indicare l'incertezza tra i k stati.

Si costruiscono poi gli archi uscenti dagli stati collettivi, nel modo seguente.

Se uno stato collettivo contiene gli stati p_1, p_2, \dots, p_k , per ognuno di essi si considerano in N gli archi uscenti, etichettati con la stessa lettera a

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

e si uniscono gli stati di arrivo

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

ottenendo lo stato collettivo di arrivo della transizione

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

Se tale stato ancora non esiste in M' , lo si crea.

Algoritmo 3.21. Algoritmo dell'insieme delle parti finite.

L'automa M' deterministico equivalente a N ha:

1. gli stati $Q' = \mathcal{P}(Q)$, l'insieme delle parti finite di Q
2. gli stati finali $F' = \{p' \in Q' \mid p' \cap F \neq \emptyset\}$, quelli contenenti uno stato finale di N
3. lo stato iniziale⁸ $[q_0]$
4. la funzione di transizione δ' :
per ogni $p' \in Q'$ e per ogni $a \in \Sigma$

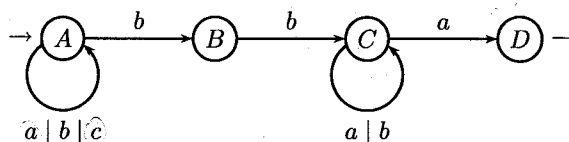
$$p' \xrightarrow{a} [s \mid q \in p' \wedge (\text{l'arco } q \xrightarrow{a} s \text{ è in } N)]$$

Al passo 4., se l'arco $q \xrightarrow{a} q_{err}$ porta nello stato d'errore, esso non va aggiunto allo stato collettivo; infatti i calcoli che portano nel pozzo non riconoscono alcuna stringa e possono essere ignorati.

Poiché gli stati di M' sono i sottoinsiemi di Q , la cardinalità di Q' è, nel caso peggiore, un esponenziale di quella di Q , a comprova della maggiore dimensione della macchina deterministica. Si ricordi l'esplosione esponenziale del numero degli stati nel linguaggio in cui il k -ultimo carattere è fissato (p. 108).

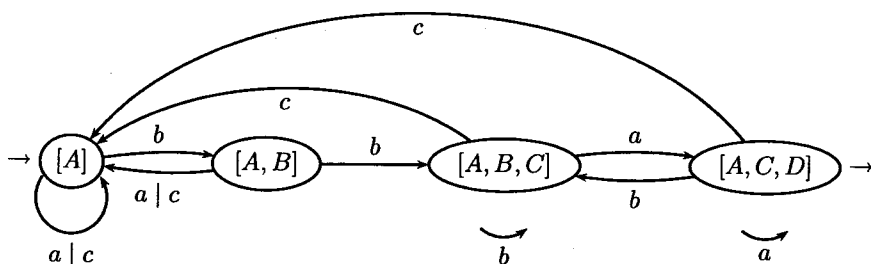
L'algoritmo può essere migliorato: M' spesso ha degli stati irraggiungibili dallo stato iniziale, dunque inutili. Invece di eliminarli, ripulendo l'automa a posteriori, conviene evitarne la creazione: si disegnano soltanto gli stati che sono raggiungibili dallo stato iniziale.

Esempio 3.22. Dato l'automa indeterministico



ecco l'automa deterministico equivalente:

⁸Se l'automa dato N ha più stati iniziali, lo stato iniziale di M' è l'insieme di tutti gli stati iniziali.



Spiegazioni: essendo $\delta(A, b) = \{A, B\}$ si crea lo stato collettivo $[A, B]$, da cui partono le transizioni

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

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

Poi si creano le transizioni uscenti dal nuovo stato collettivo $[A, B, C]$:

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

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

ecc.

Si termina quando il passo 4, applicato agli stati di Q' finora creati, non crea nuovi stati.

Si noti che non tutti i sottoinsiemi di Q corrispondono a stati raggiungibili: ad es. lo stato $[A, C]$ è inutile.

Per giustificare la validità del procedimento, si dimostra che una stringa x è riconosciuta da M' se, e solo se, è accettata da M .

Se un calcolo di M accetta x , esiste un cammino etichettato x dallo stato iniziale q_0 a uno stato finale q_f . L'algoritmo garantisce allora che anche in M' esiste un cammino etichettato x da $[q_0]$ a uno stato $[\dots, q_f, \dots]$ contenente q_f . Viceversa se x è l'etichetta d'un calcolo valido di M' , da $[q_0]$ a uno stato finale $p \in F'$, allora per costruzione p contiene almeno uno stato finale q_f di M . Per costruzione esiste allora anche in M un cammino di etichetta x da q_0 a q_f . In conclusione vale la seguente proprietà fondamentale.

Proprietà 3.23. Ogni linguaggio a stati finiti è riconosciuto da un automa deterministico.

Questa proprietà testimonia che l'algoritmo di riconoscimento dei linguaggi a stati finiti opera in tempo reale, ossia richiede un numero di transizioni eguale o minore del numero dei caratteri della stringa (minore se un errore si manifesta prima del completamento della lettura della stringa).

Come corollario si ha che, per ogni linguaggio riconosciuto da un automa a stati finiti, esiste una grammatica unilineare priva di ambiguità, quella che

corrisponde in modo naturale (p. 106) all'automa deterministico. Di conseguenza per i linguaggi regolari si dispone d'un procedimento per eliminare l'ambiguità, attraverso la costruzione del riconoscitore deterministico e della grammatica a esso equivalente.

✚ 3.8 Dall'espressione regolare all'automa riconoscitore

Se per definire un linguaggio si usa un'espressione regolare, invece d'una grammatica unilineare, resta da esporre un procedimento per costruire l'automa. Il problema è assai diffuso nelle applicazioni, come la compilazione o l'analisi dei documenti, e tanti algoritmi sono stati inventati. Essi si diversificano rispetto alle proprietà (deterministico o indeterministico con o senza mosse spontanee) e alle dimensioni dell'automa prodotto, e rispetto alla complessità algoritmica della costruzione.

Si esporranno due metodi di costruzione. Il primo, noto come metodo di Thompson o strutturale, decompone l'espressione nelle sue sottoespressioni, fino a giungere agli elementi atomici dell'alfabeto. Costruisce poi i riconoscitori delle varie sottoespressioni e li connette in reti di riconoscitori che realizzano gli operatori (unione, concatenamento, stella, croce) presenti nella e.r..

Il secondo metodo, noto come algoritmo di Glushkov, McNaughton e Yamada, costruisce un riconoscitore indeterministico senza mosse spontanee, ma di dimensioni maggiori di quello costruito col metodo precedente.

Con questi metodi si concluderà il percorso espositivo che mostra che le famiglie dei linguaggi regolari, a stati finiti e unilineari (tipo 3 di Chomsky) sono identiche.

Entrambi i metodi possono essere combinati con gli algoritmi di determinazione allo scopo di produrre direttamente macchine deterministiche.

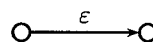
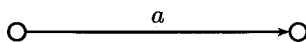
✚ 3.8.1 Metodo strutturale o di Thompson

Sia data un'espressione regolare; analizzandola nelle sottoespressioni, si costruirà ora in modo sistematico l'automa che riconosce il linguaggio.

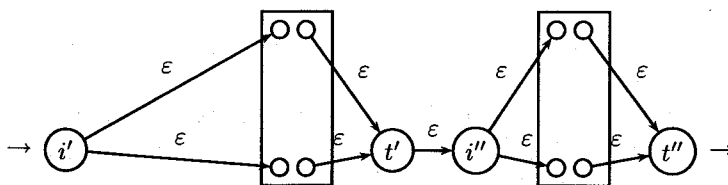
In questa costruzione ogni automa deve avere un solo stato iniziale e un solo stato finale. Se così non fosse, si aggiunge il nuovo stato iniziale i e lo si collega ai vecchi stati iniziali tramite ϵ -archi. Dualmente, se vi fossero più stati finali, si introduce un nuovo stato finale t , e i vecchi stati finali sono collegati al nuovo stato tramite ϵ -archi.

Il procedimento di Thompson [49] sfrutta le regole di corrispondenza tra e.r. e automi riconoscitori di seguito schematizzate.

Espressioni regolari atomiche:

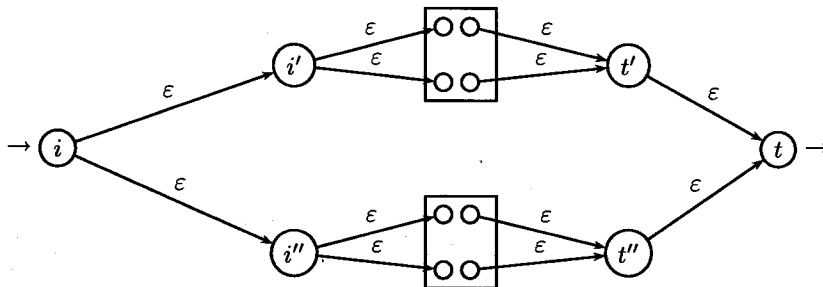


Concatenamento di due e.r.:

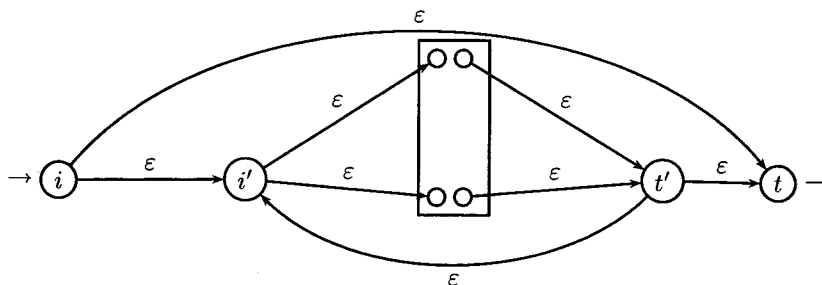


dove il rettangolo schematizza un automa di cui si evidenziano soltanto gli stati iniziali (colonna di sinistra) e finali (colonna di destra).

Unione di due e.r.:



Stella d'una e.r.:



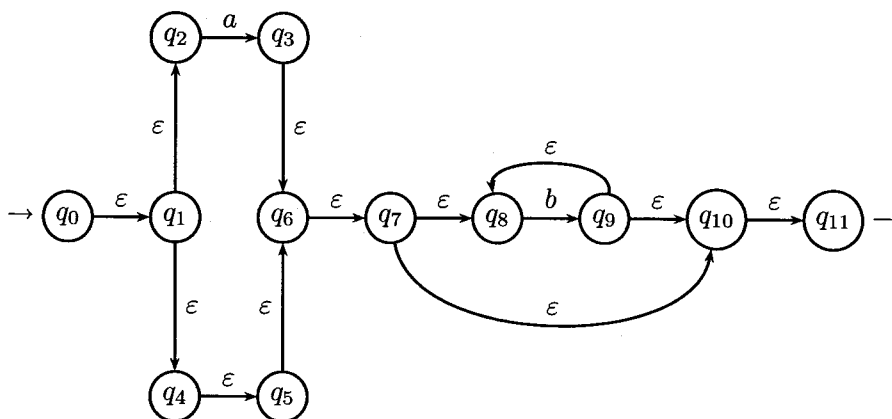
In generale il risultato della costruzione è un automa indeterministico, con archi spontanei.

Per giustificare il procedimento di Thompson, basta osservare che esso è una formulazione operativa delle proprietà di chiusura dei linguaggi regolari rispetto alle operazioni di concatenamento, d'unione e di stella, enunciate nel capitolo precedente (p. 24).

Esempio 3.24. Per l'e.r.:

$$(a \cup \epsilon).b^*$$

si disegna l'automa costruito con il metodo strutturale:



Si noti che diversi stati (ad es. q_0 e q_{11}) sono ridondanti; potrebbero essere fusi con q_1 e q_{10} rispettivamente. Esistono versioni perfezionate dell'algoritmo di Thompson che evitano tali ridondanze durante la costruzione.

Altre versioni abbinano l'algoritmo con quello per la eliminazione delle mosse spontanee. Volendo realizzare con un programma questo procedimento, è necessario decomporre una e.r. nelle sue sottoespressioni, produrre i riconoscitori dei linguaggio ad esse associati, e poi collegarli mediante le mosse spontanee. Questo è un tipico problema di analisi e traduzione guidata dalla sintassi o struttura della e.r., affrontabile con i metodi del capitolo 5.

3.8.2 Algoritmo di Glushkov, Mc Naughton e Yamada

Un altro metodo classico, GMY, costruisce un automa i cui stati sono in corrispondenza diretta con i caratteri terminali presenti nell'espressione. È necessario preparare il terreno⁹ con la definizione d'una particolare sottofamiglia dei linguaggi regolari.

Linguaggi localmente testabili e automi locali

Vi sono linguaggi regolari semplicissimi da riconoscere, perché basta controllare la presenza di certe sottostringhe. Un esempio è l'insieme delle stringhe che iniziano con b , finiscono con a o b , e contengono le sottostringhe ba, ab .

Definizione 3.25. Dato un linguaggio L di alfabeto Σ ,
l'insieme degli inizi è

$$\text{Ini}(L) = \{a \in \Sigma \mid a\Sigma^* \cap L \neq \emptyset\} \quad (\text{ossia le lettere iniziali di } L)$$

l'insieme delle fini è

$$\text{Fin}(L) = \{a \in \Sigma \mid \Sigma^*a \cap L \neq \emptyset\} \quad (\text{ossia le lettere finali di } L)$$

l'insieme dei digrammi è

$$\text{Dig}(L) = \{x \in \Sigma^2 \mid \Sigma^*x\Sigma^* \cap L \neq \emptyset\} \quad (\text{le sottostringhe di lunghezza 2})$$

e il suo complemento

$$\overline{\text{Dig}}(L) = \Sigma^2 \setminus \text{Dig}(L)$$

Esempio 3.26. Linguaggio localmente testabile.

Per il linguaggio $L_1 = (abc)^+$ si ha

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

e quindi

$$\overline{\text{Dig}}(L_1) = \{aa, ac, ba, bb, cb, cc\}$$

⁹Seguendo il percorso concettuale di Berstel e Pin [10].

Si osservi che i tre insiemi caratterizzano esattamente le frasi del linguaggio, nel senso che vale l'identità

$$L_1 \equiv \{x \mid \text{Ini}(x) \in \text{Ini}(L_1) \wedge \text{Fin}(x) \in \text{Fin}(L_1) \wedge \text{Dig}(x) \subseteq \text{Dig}(L_1)\} \quad (3.1)$$

o equivalentemente

$$L_1 \equiv \{x \mid \text{Ini}(x) \in \text{Ini}(L_1) \wedge \text{Fin}(x) \in \text{Fin}(L_1)\} \setminus \Sigma^* \overline{\text{Dig}}(L_1) \Sigma^* \quad (3.2)$$

Definizione 3.27. *Un linguaggio L è detto locale (o localmente testabile) se per esso vale l'identità 3.1 (o 3.2).*

Chiaramente per qualsiasi linguaggio, purché non contenente la stringa vuota, la formula 3.1 (o 3.2) vale sempre, pur di sostituire il segno di inclusione a quello di identità: infatti ogni frase inizia (risp. termina) necessariamente con un carattere di *Ini* (risp. di *Fin*) e i suoi digrammi sono inclusi in quelli del linguaggio. Ma tali condizioni possono essere insufficiente per escludere alcune stringhe non valide.

Esempio 3.28. Linguaggio non locale.

Nel linguaggio $L_2 = b(aa)^+b$ si ha

$$\text{Ini}(L_2) = \text{Fin}(L_2) = \{b\} \quad \text{Dig}(L_2) = \{aa, ab, ba\} \quad \overline{\text{Dig}}(L_2) = \{bb\}$$

Le frasi di L_2 hanno lunghezza pari, mentre l'insieme delle frasi, che iniziano e terminano con b e non contengono il digramma bb , comprende anche stringhe di lunghezza dispari, come $baaab$. Quindi tale linguaggio include strettamente L_2 .

Il riconoscimento d'un linguaggio locale è particolarmente semplice. Scandendo la stringa da sinistra a destra, si controlla che il primo carattere stia in *Ini*, ogni coppia di caratteri adiacenti non stia in $\overline{\text{Dig}}$ e l'ultimo carattere stia in *Fin*. Il controllo può essere effettuato facendo scorrere una finestra mobile, larga due posizioni, da sinistra a destra sulla stringa, operazione facilmente attuabile da un automa finito deterministico.

Algoritmo di costruzione del riconoscente d'un linguaggio locale

L'automa riconoscente d'un linguaggio locale, definito dagli insiemi *Ini*, *Fin*, *Dig*, è definito da:

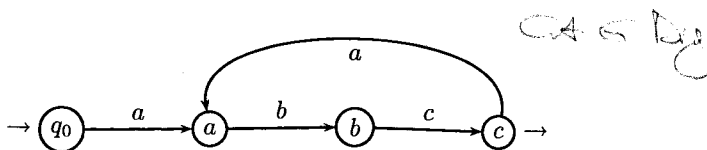
- gli stati $q_0 \cup \Sigma$;
- gli stati finali *Fin*;
- gli archi $q_0 \xrightarrow{a} a$ se $a \in \text{Ini}$;
- gli archi $a \xrightarrow{b} b$ se $ab \in \text{Dig}$;
- se la stringa vuota appartiene al linguaggio, lo stato iniziale q_0 è anche finale.

Le seguenti proprietà caratterizzano i riconoscenti dei linguaggi locali:

1. gli stati non iniziali sono in corrispondenza biunivoca con l'alfabeto
2. nessuna freccia entra nello stato iniziale q_0
3. tutte le frecce etichettate con la stessa lettera a entrano nello stesso stato, quello denominato a .

Intuitivamente, l'automa si trova nello stato b se, e solo se, l'ultimo carattere letto è b .

Esempio 3.29. Ecco il riconoscitore del linguaggio locale $(abc)^+$ dell'es. 3.26:



Grazie alle proprietà elencate, le etichette delle frecce sono pleonastiche, perché sono identiche a quelle dei nodi di destinazione.

Composizione di linguaggi locali di alfabeti disgiunti

Per applicare l'idea della finestra scorrevole a e.r. generiche, occorre un altro passo concettuale, basato sulla seguente osservazione: le operazioni fondamentali preservano la proprietà dei linguaggi locali, a condizione che i loro alfabeti terminali siano disgiunti.

Proprietà 3.30. Dati i linguaggi locali L' e L'' , di alfabeti disgiunti, $\Sigma' \cap \Sigma'' = \emptyset$, risultano locali anche i linguaggi unione $L' \cup L''$, concatenamento $L'.L''$ e stella L'^* (e anche croce).

Dimostrazione: è immediato costruire per il linguaggio risultante un riconoscitore del tipo locale, combinando quelli dei linguaggi componenti. Siano q'_0, q''_0 i loro rispettivi stati iniziali e F', F'' gli insiemi degli stati finali.

Per l'unione $L' \cup L''$:

- lo stato iniziale q_0 è ottenuto fondendo insieme gli stati iniziali q'_0 e q''_0 ;
- gli stati finali sono quelli dei due automi, $F' \cup F''$;
si noti che, se la stringa vuota appartiene a (almeno) uno dei due linguaggi, si aggiunge q_0 agli stati finali.

Per il concatenamento $L'.L''$:

- lo stato iniziale è q'_0
- gli archi sono quelli di L' uniti a:
 - gli archi di L'' , tranne quelli aventi origine nello stato iniziale q''_0 ;

- in sostituzione di questi ultimi, gli archi $q' \xrightarrow{a} q''$, aventi origine in uno stato finale $q' \in F'$ e destinazione in uno stato q'' , tali che esista per L'' l'arco $q_0' \xrightarrow{a} q''$.
- gli stati finali sono quelli di F'' , se $\varepsilon \notin L''$;
gli stati finali F' uniti a F'' , altrimenti.

Per la stella L'^* :

- si aggiunge q_0' agli stati finali;
- per ogni stato finale $q \in F'$, si aggiunge l'arco $q \xrightarrow{a} r$, se l'automa aveva l'arco $q_0' \xrightarrow{a} r$, con origine nello stato iniziale.

Questa dimostrazione è costruttiva e produce il riconoscitore d'un linguaggio ottenuto unendo, concatenando o iterando dei linguaggi locali, purché di alfabeti disgiunti.

Procedimento GMY

Si vedrà ora come trasformare una generica e.r. in un linguaggio locale, attraverso un opportuno cambiamento di alfabeto.

In una e.r. un carattere può ovviamente comparire più volte. La e.r. è detta *lineare* se nessun carattere è ripetuto. Ad es. $(abc)^*$ è lineare, mentre $(ab)^*a$ non è lineare, perché il carattere a compare più volte (pur essendo locale il linguaggio).

Proprietà 3.31. Il linguaggio definito da una e.r. lineare è locale.

Dimostrazione. Per l'ipotesi di linearità, le sottoespressioni della e.r. sono di alfabeti disgiunti. Poiché la e.r. è ottenuta per composizione dei linguaggi locali associati alle sue sottoespressioni, il linguaggio da essa definito è locale, grazie alla Proprietà 3.30.

Esempio 3.32. Per la e.r. lineare $(ab \cup c)^*$ seguono i passi della costruzione del riconoscitore, partendo dalle sottoespressioni atomiche:



sottoespressione	automi componenti
elementi a, b, c \cdot	
concatenamento e unione $ab \cup c$	
stella $(ab \cup c)^*$	

Avendo accertato che il linguaggio d'una e.r. lineare è locale, il problema di costruirne il riconoscitore si riconduce al calcolo degli insiemi dei caratteri iniziali Ini , finali Fin e dei digrammi Dig del linguaggio.

Calcolo degli insiemi locali d'un linguaggio regolare

Date le e.r. e, e' , le seguenti regole, da applicare ai caratteri terminali e agli operatori, calcolano i tre insiemi.

È necessario prima accertare se la e.r. definisce anche la stringa vuota, nel qual caso essa è detta *annullabile*. La funzione $Null(e)$ assume il valore ε , se $\varepsilon \in L(e)$; altrimenti assume il valore \emptyset . Essa è calcolata mediante le regole:

$$\begin{array}{ll} Null(\varepsilon) = \varepsilon & Null(\emptyset) = \emptyset \\ Null(a) = \emptyset, \text{ per ogni terminale } a & Null(e \cup e') = Null(e) \cup Null(e') \\ Null(e.e') = Null(e) \cap Null(e') & Null(e^*) = \varepsilon \\ Null(e^+) = Null(e) & \end{array}$$

Ad es. si ha:

$$Null((a \cup b)^* ba) = Null((a \cup b)^*) \cap Null(ba) = \varepsilon \cap (Null(b) \cap Null(a)) = \varepsilon \cap \emptyset = \emptyset$$

Le seguenti regole calcolano i tre insiemi locali:

<i>Inizi</i>	<i>Fini</i>
$Ini(\emptyset) = \emptyset$	$Fin(\emptyset) = \emptyset$
$Ini(\varepsilon) = \emptyset$	$Fin(\varepsilon) = \emptyset$
$Ini(a) = \{a\}$, per ogni terminale a	$Fin(a) = \{a\}$, per ogni terminale a
$Ini(e \cup e') = Ini(e) \cup Ini(e')$	$Fin(e \cup e') = Fin(e) \cup Fin(e')$
$Ini(e.e') = Ini(e) \cup Null(e)Ini(e')$	$Fin(e.e') = Fin(e') \cup Fin(e)Null(e')$
$Ini(e^*) = Ini(e^+) = Ini(e)$	$Fin(e^*) = Fin(e^+) = Fin(e)$

Nota: le regole per il calcolo di *Ini* e di *Fin* sono duali, nel senso che coincidono se si sostituisce alla e.r. la sua riflessa.

Digrammi

$$\begin{array}{l} \underline{Dig(\emptyset) = \emptyset} \\ Dig(\varepsilon) = \emptyset \\ Dig(a) = \emptyset, \text{ per ogni terminale } a \\ Dig(e \cup e') = Dig(e) \cup Dig(e') \\ Dig(e.e') = Dig(e) \cup Dig(e') \cup Fin(e)Ini(e') \\ Dig(e^*) = Dig(e^+) = Dig(e) \cup Fin(e)Ini(e) \end{array}$$

Terminati questi calcoli, si può costruire il riconoscitore d'una e.r. lineare, come sotto illustrato.

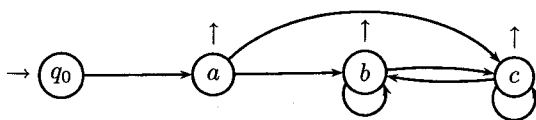
Esempio 3.33. Riconoscitore di e.r. lineare.

La e.r. $a(b \cup c)^*$ non è annullabile, e si pone $Null = \emptyset$.

Calcolati gli insiemi locali

$$Ini = a \quad Fin = \{b, c\} \cup a = \{a, b, c\} \quad Dig = \{ab, ac\} \cup \{bb, bc, cb, cc\}$$

si costruisce, con l'algoritmo di p. 127, l'automa locale:



dove al solito le etichette degli archi sono il nome dello stato d'arrivo.

Numerazione della espressione regolare

Per consentire l'applicazione del procedimento a una e.r. generica, la si trasforma in una e.r. lineare sostituendo i caratteri terminali con altri tutti distinti, ottenuti numerando progressivamente i caratteri della e.r..

Così l'espressione

$$(ab)^*a \text{ diventa } (a_1b_2)^*a_3$$

La seconda e.r. è lineare nell'alfabeto dei caratteri numerati, e quindi definisce un linguaggio locale.

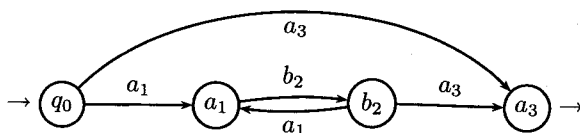
Costruito nel modo noto il riconoscitore della e.r. numerata, quello della e.r. originale si ottiene infine, cancellando i numeri dalle etichette degli archi.¹⁰ Conviene riepilogare i passi dell'algoritmo.

Algoritmo 3.34. GMY

L'algoritmo si articola in quattro passi:

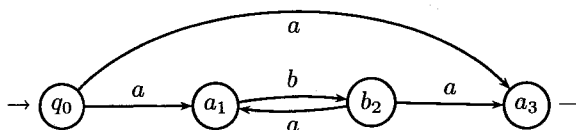
1. Numera l'e.r. e ottenendo una e.r. lineare e' ;
2. Calcola per e' l'annullabilità e gli insiemi locali Ini, Fin, Dig ;
3. Costruisci il riconoscitore del linguaggio locale caratterizzato da tali insiemi (come spiegato a p. 127);
4. Cancella la numerazione dalle etichette degli archi.

Esempio 3.35. Per la e.r. $(ab)^*a$ si calcola la e.r. numerata $(a_1b_2)^*a_3$ e se ne costruisce il riconoscitore:



Cancellando i numeri dagli archi, si ottiene il riconoscitore del linguaggio desiderato:

¹⁰Questo è un esempio di traslitterazione (omomorfismo) definito a p. 80.



Si osservi che l'automata è indeterministico, privo di mosse spontanee, e ha tanti stati quanti i caratteri terminali presenti nella e.r., più uno.

3.8.3 Costruzione del riconoscitore deterministico di Berry e Sethi

Data una e.r., per ottenere l'automata deterministico che la riconosce, si potrebbe trasformare l'automata costruito da GMY, mediante l'algoritmo delle parti finite; ma conviene presentare un algoritmo diretto.¹¹

Sia e la e.r. di alfabeto Σ e sia e' quella numerata, di alfabeto Σ_N , con gli insiemi locali *Ini*, *Fin*, *Dig* e la funzione *Null*.

Per convenienza, invece dei digrammi, si usa l'insieme dei *séguiti* d'un carattere nella e.r. numerata, così definito:

$$Seg(a_i) = \{b_j \mid a_i b_j \in Dig(e')\}$$

Inoltre, per convenzione, lo speciale carattere \dashv (pensabile come terminatore della stringa) sta nei *séguiti* dei caratteri finali:

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

(ciò equivale a considerare la e.r. $e' \dashv$ invece di e'). Il terminatore non ha seguito, $Seg(\dashv) = \emptyset$.

Algoritmo 3.36. Algoritmo BS (Berry e Sethi)

Ogni stato dell'automata è denotato da un sottoinsieme di $\Sigma_N \cup \dashv$.

L'algoritmo esamina ogni stato, per costruire gli archi uscenti e i loro stati di arrivo, applicando una regola simile a quella dell'algoritmo delle parti finite. Dopo l'esame, lo stato è marcato come visitato, per evitare di riesaminarlo.

Lo stato iniziale è $Ini(e' \dashv)$. Uno stato è finale se contiene \dashv .

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 carattere $b \in \Sigma$

 do

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

 se q' non è vuoto né appartiene a Q ,

 aggiungilo come nuovo stato non visitato, ponendo

¹¹Per approfondimenti si veda [7].

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

aggiungi l'arco $q \xrightarrow{b} q'$

end do

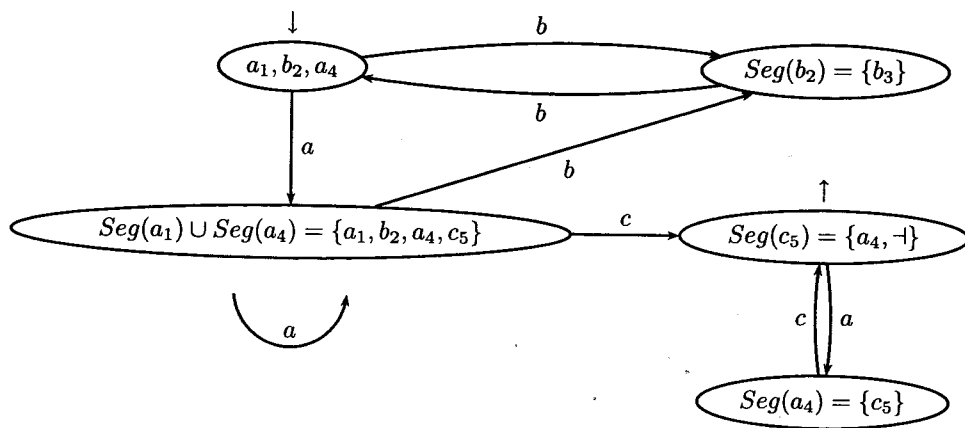
end do

Esempio 3.37. Data l'e.r.

$(a \mid bb)^*(ac)^+$, numerata $(a_1 \mid b_2b_3)^*(a_4c_5)^+ \neg$, si calcola l'insieme $Ini(e') = \{a_1, b_2, a_4\}$, poi la funzione *seguiti*

	<i>seguiti</i>
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, \neg

e infine l'automa deterministico:



Gli automi costruiti con questi procedimenti contengono spesso più stati del necessario, e, se necessario, possono essere minimizzati nella maniera nota.

Applicazione alla determinizzazione d'un automa

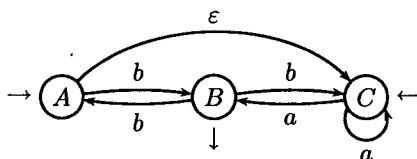
L'algoritmo BS può essere usato, in alternativa a quello delle parti finite, per costruire un automa deterministico M equivalente a un automa indeterministico N , che può contenere ε -archi.

Si procede nel modo seguente.

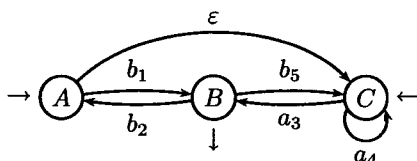
1. Si numerano in modo distinto le etichette degli archi non spontanei di N . L'automa numerato N' ottenuto riconosce un linguaggio locale.
2. Si calcolano gli insiemi locali Ini , Fin , $Seguiti$ per l'automa N' . Il calcolo si svolge con regole analoghe a quelle usate per una e.r..
3. Si applica la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

È sufficiente un esempio per illustrare l'applicazione.

Esempio 3.38. Dato l'automa indeterministico N



si numerano le etichette terminali, ottenendo l'automa N' :



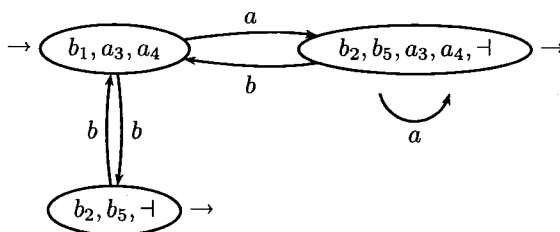
il cui linguaggio è locale. La stringa vuota non appartiene al linguaggio. Si calcola l'insieme degli inizi

$$Ini(L(N') \neg) = \{b_1, a_3, a_4\}$$

notando che è $\varepsilon a_4 = a_4$, $\varepsilon a_3 = a_3$; poi l'insieme dei seguiti

	seguiti
b_1	b_2, b_5, \neg
b_2	b_1, a_3, a_4
a_3	b_2, b_5, \neg
a_4	a_3, a_4
b_5	a_3, a_4

Infine l'algoritmo BS costruisce l'automa deterministico M :



3.9 Espressioni regolari con complemento e intersezione

Lo studio delle operazioni sui linguaggi regolari è qui completato considerando il complemento, l'intersezione e la differenza insiemistica, che erano stati lasciati in sospeso. Infatti si incontrano situazioni in cui queste operazioni rendono più facile o più concisa l'espressione del linguaggio.

Proprietà 3.39. Chiusura di *REG* rispetto a complemento e intersezione.

Siano L e L' linguaggi regolari. Allora anche il complemento $\neg L$ e l'intersezione $L \cap L'$ sono linguaggi regolari.

Si inizia dalla costruzione del riconoscitore del complemento $\neg L = \Sigma^* \setminus L$. Si può supporre che il riconoscitore M di L sia deterministico, con stato iniziale q_0 , stati Q , stati finali F , e con funzione di transizione δ .

Algoritmo 3.40. Costruzione del riconoscitore deterministico \overline{M} del complemento.

Per primo si completa l'automa M , aggiungendo lo stato d'errore o pozzo p e gli archi che in esso entrano:

1. Crea un nuovo stato $p \notin Q$, il pozzo; gli stati di \overline{M} sono $Q \cup \{p\}$
2. la funzione di transizione $\overline{\delta}$ è:
 - a) $\overline{\delta}(q, a) = \delta(q, a)$, dove $\delta(q, a) \in Q$;
 - b) $\overline{\delta}(q, a) = p$, dove $\delta(q, a)$ non è definita;
 - c) $\overline{\delta}(p, a) = p$, per ogni carattere $a \in \Sigma$
3. gli stati finali sono $\overline{F} = (Q \setminus F) \cup \{p\}$.

Si noti che gli stati finali e non finali sono stati scambiati.

Per primo si osservi che, se un calcolo di M riconosce la stringa $x \in L(M)$, allora il corrispondente calcolo di \overline{M} termina in uno stato non finale, ossia $x \notin L(\overline{M})$.

Poi, 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, ossia $y \in L(\overline{M})$.

Da ultimo, per dimostrare che l'intersezione di due linguaggi regolari è regolare, basta invocare la nota identità di De Morgan:

$$L \cap L' = \neg(\neg L \cup \neg L')$$

poiché, sapendo che i linguaggi $\neg L$ e $\neg L'$ sono regolari, la loro unione è regolare.

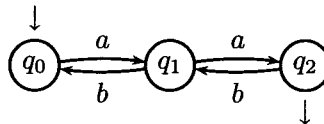
Come corollario, anche la *differenza insiemistica* di due linguaggi regolari è regolare grazie all'identità

$$L \setminus L' = L \cap \neg L'$$

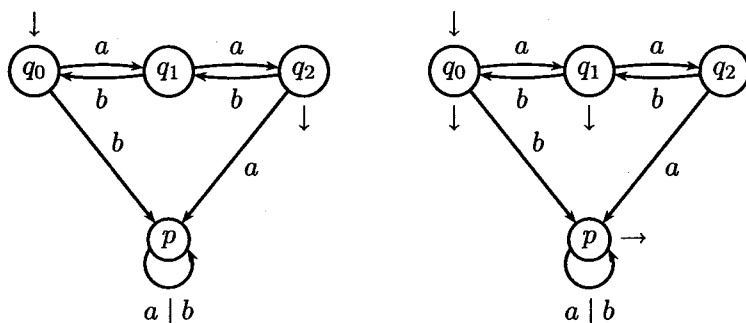
Esempio 3.41. Automa del complemento.

La seguente figura mostra l'automa dato M , quello intermedio completato con il pozzo, e il riconoscitore \overline{M} del complemento.

automa originale M

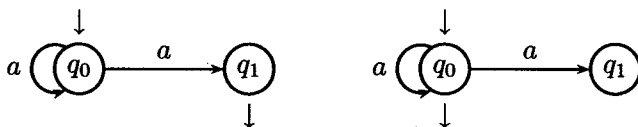


dopo il completamento con il pozzo automa \overline{M} del complemento



Per la validità del risultato, è essenziale che l'automa da trasformare sia deterministico, altrimenti il linguaggio accettato dall'automa costruito potrebbe non essere disgiunto da quello originale, violando la proprietà del complemento, $L \cap \neg L = \emptyset$. Come controesempio, si veda

automa originale pseudo-automa del complemento



dove l'automa dello pseudo-complemento accetta erroneamente la stringa a , appartenente al linguaggio originale.

Per ultimo osserviamo che la costruzione può produrre automi che hanno stati inutili, o comunque automi non minimi.

3.9.1 Prodotto di automi

Una manipolazione frequente nelle applicazioni della teoria degli automi è il *prodotto (cartesiano)*, che consiste nella simulazione di due automi mediante un unico automa, avente come insieme di stati il prodotto cartesiano degli stati delle due macchine. Se ne vedrà ora l'impiego per costruire il riconoscitore del linguaggio intersezione di due linguaggi regolari.

Per inciso, la dimostrazione della chiusura della famiglia *REG* rispetto all'intersezione, basata sull'identità di De Morgan (p. 137), già fornisce un procedimento per riconoscere l'intersezione: dati i riconoscitori finiti deterministici di

due linguaggi, si costruiscono quelli dei loro complementi, e poi il riconoscitore dell'unione, (applicando il metodo di Thompson di p. 123). Da quest'ultimo (dopo averlo reso deterministico) si costruisce infine l'automa del complemento.

In modo più diretto, l'intersezione di due linguaggi regolari può essere riconosciuta dalla macchina prodotto cartesiano dei due automi dati M' e M'' . Per semplicità si suppone che essi 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 $Q' \times Q''$. Pertanto ogni stato è una coppia $\langle q', q'' \rangle$, dove la prima (seconda) componente è uno stato della prima (seconda) macchina.

In tale stato si definisce l'arco uscente

$$\langle q', q'' \rangle \xrightarrow{a} \langle r', r'' \rangle$$

se, e solo se, esistono gli archi $q' \xrightarrow{a} r'$ in M' e $q'' \xrightarrow{a} r''$ in M'' .

Ossia tale arco sta in M se, e solo se, la sua proiezione sulla prima (risp. seconda) componente sta in M' (risp. in M'').

Gli stati iniziali I di M sono il prodotto $I = I' \times I''$ degli stati iniziali delle macchine componenti; e quelli finali sono il prodotto degli stati finali, $F = F' \times F''$. Per giustificare la validità della costruzione, si consideri una stringa x appartenente all'intersezione: essa è accettata da un calcolo di M' e da un calcolo di M'' , dunque anche da un calcolo di M che passa attraverso le coppie di stati, rispettivamente visitati dai due calcoli.

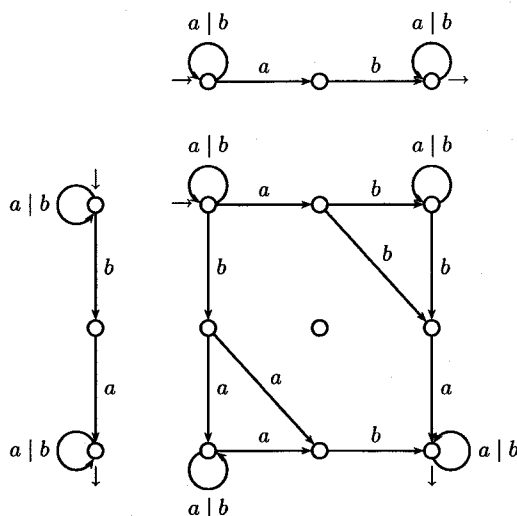
Viceversa se x non sta nell'intersezione, almeno uno dei calcoli di M' e M'' non raggiunge uno stato finale, dunque neanche il calcolo di M raggiunge uno stato finale.

Esempio 3.42. Intersezione e macchina prodotto (Sakarovitch).

Il riconoscitore delle stringhe che contengono almeno una sottostringa ab e una sottostringa ba è specificato molto naturalmente mediante l'intersezione dei linguaggi L', L'' seguenti

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

e quindi il prodotto cartesiano dei riconoscitori dei due linguaggi:



Le coppie del prodotto cartesiano non raggiungibili dallo stato iniziale possono al solito essere omesse.

Il metodo del prodotto cartesiano può essere applicato anche a altre operazioni diverse dall'intersezione; così, nel caso dell'unione, sarebbe facile modificare la macchina prodotto in modo che essa riconosca una stringa, se almeno una delle macchine componenti la riconosce. Tuttavia la macchina prodotto ha un numero di stati maggiore dell'automa, costruito con il metodo strutturale di Thompson: il prodotto delle cardinalità invece della somma.

Espressioni regolari estese e ristrette

Una *espressione regolare* è detta *estesa* se contiene, in aggiunta alle operazioni di base (unione, concatenamento e stella), anche il complemento, l'intersezione o la differenza insiemistica.

Ad es. le stringhe, che contengono almeno una sottostringa aa e non terminano con bb , sono definite dalla e.r. estesa

$$((a|b)^*aa(a|b)^*) \cap \neg((a|b)^*bb)$$

Segue un esempio più applicativo, per dimostrare l'utilità delle e.r. estese.

Esempio 3.43. Identificatori.

Gli identificatori validi possono contenere le lettere $a \dots z$, le cifre $0 \dots 9$ (non in prima posizione) e il trattino $'-'$ (non in prima né in ultima posizione). Non sono permessi due trattini consecutivi.

Una frase è: *dopo - 2ndo - test.*

La seguente e.r. estesa prescrive che le stringhe (i) inizino con una lettera, (ii) non contengano due tratti consecutivi e (iii) non terminino con un tratto:

$$\begin{aligned}
 & \underbrace{(a \dots z)^+ (a \dots z \mid 0 \dots 9 \mid -)^*}_{(i)} \cap \\
 & \underbrace{\neg((a \dots z \mid 0 \dots 9 \mid -)^* - (a \dots z \mid 0 \dots 9 \mid -)^*)}_{(ii)} \cap \\
 & \underbrace{\neg((a \dots z \mid 0 \dots 9 \mid -)^* -)}_{(iii)}
 \end{aligned}$$

Linguaggi senza stella

L'uso degli operatori di complemento e di intersezione non aggiunge nulla alla famiglia dei linguaggi regolari, perché questi operatori possono essere eliminati riducendoli a quelli di base (unione, concatenamento, stella).

Invece, l'eliminazione della stella dagli operatori permessi causa una perdita nella famiglia dei linguaggi definibili, che costituiscono una sottoclasse di *REG*, detta la famiglia dei linguaggi *aperiodici* o *senza stella* (star free) o *senza contatore* (non-counting).

Poiché tale sottofamiglia è raramente considerata nell'ambito della compilazione, basti qui un cenno.

Si considerino gli operatori d'unione, concatenamento, intersezione e complemento. Partendo dagli elementi terminali dell'alfabeto e dall'insieme vuoto, si può allora definire una famiglia di linguaggi mediante delle *espressioni regolari senza stella*, facenti uso esclusivo di detti operatori. Tali e.r. differiscono da quelle classiche perché possono contenere l'intersezione e il complemento, ma non la stella (né la croce).

Un linguaggio è detto *senza stella* se esiste una e.r. senza stella che lo definisce.

Si nota subito che il linguaggio universale di alfabeto Σ , essendo il complemento dell'insieme vuoto, $\Sigma^* = -\emptyset$, è un linguaggio senza stella.

In effetti, si è già incontrata, senza saperlo, una famiglia di linguaggi senza stella: i linguaggi locali (p. 126) sono infatti sempre definibili senza fare uso della stella. Si ricorda che un linguaggio locale è definito da tre insiemi caratteristici: gli inizi, le fini e i digrammi permessi (o vietati). Tale proprietà è facilmente espressa dalla intersezione di tre linguaggi senza stella.

Esempio 3.44. E.r. senza stella di un linguaggio locale.

Le frasi del linguaggio locale $(abc)^+$ dell'esempio di p. 126, iniziano con a , finiscono con c e non contengono i digrammi $\{aa \mid ac \mid ba \mid bb \mid cb \mid cc\}$. Il linguaggio è allora definito dalla e.r. senza stella:

$$(a - \emptyset) \cap (-\emptyset c) \cap (\neg(-\emptyset (aa \mid ac \mid ba \mid bb \mid cb \mid cc) - \emptyset))$$

La famiglia dei linguaggi senza stella è strettamente inclusa in quella dei linguaggi regolari. Non ne fanno parte i linguaggi caratterizzati da una proprietà

di conteggio che giustifica l'altro nome "senza contatore" della famiglia. Fuoriesce dalla famiglia senza stella il linguaggio regolare

$$\{x \in \{a \mid b\}^* \mid |x|_a \text{ è pari}\}$$

facilmente riconosciuto da un automa finito, avente un ciclo che conta modulo 2 le a incontrate. Per questo linguaggio non esiste una e.r. che non faccia uso della stella o della croce.

Nella casistica dei linguaggi inventati dall'uomo, l'operazione di conteggiare (nel senso intuitivamente spiegato) certe lettere o sottostringhe non sembra essere mai richiesta, al fine di controllare la correttezza. Per ragioni, che forse hanno a che fare con l'organizzazione della mente umana o forse con la robustezza della comunicazione, nessun linguaggio tecnico discrimina le frasi valide da quelle scorrette, sulla base d'una congruenza aritmetica ossia d'un conteggio ciclico. Si immagini la stranezza d'un linguaggio di programmazione, in cui la validità d'un blocco di istruzioni dipendesse dalla classe di congruenza del numero delle istruzioni presenti: blocco valido se il numero è multiplo pongasi di tre, scorretto altrimenti.

Nello studio dei linguaggi regolari utilizzati nella comunicazione uomo-macchina, sarebbe dunque sufficiente restringere l'attenzione ai linguaggi senza stella o contatore ciclico.

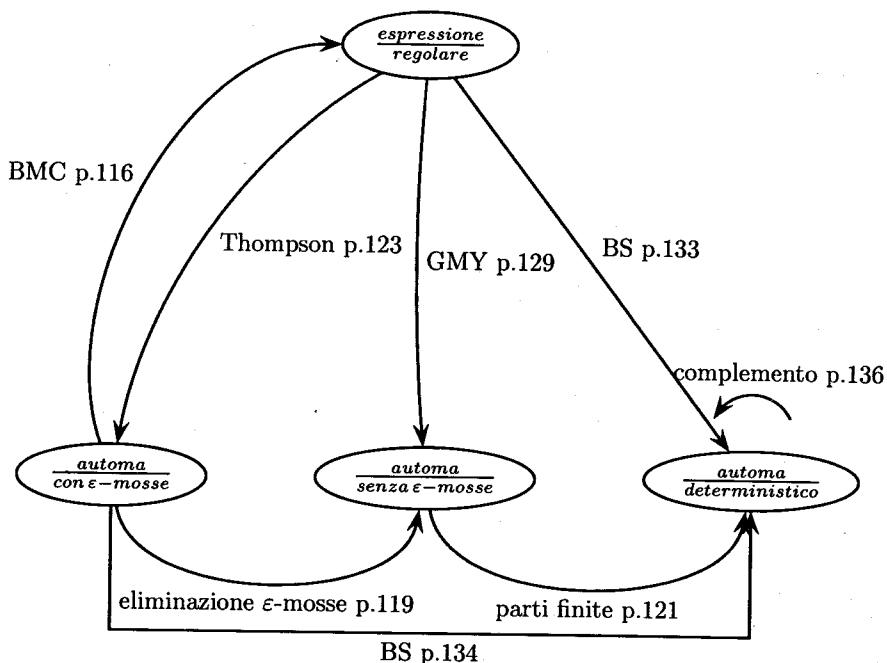
Ma ciò non vale in altri ambiti dell'informatica, anzi è ovvio che molti circuiti logici dei microprocessori svolgono operazioni di conteggio; basti pensare che uno dei più semplici componenti, il circuito bistabile o flip-flop, conta modulo due gli impulsi, ossia riconosce linguaggio $(11)^*$, che non è senza stella.¹²

3.10 Riepilogo: relazioni tra linguaggi regolari, automi e grammatiche

A conclusione dello studio dei linguaggi regolari e degli automi finiti, è utile ricapitolare le relazioni e le trasformazioni tra i modelli formali di questa famiglia di linguaggi.

La prossima figura delinea, sotto forma di grafo di flusso, i procedimenti che trasformano espressioni regolari in automi e viceversa, nonché le conversioni tra vari tipi di automi:

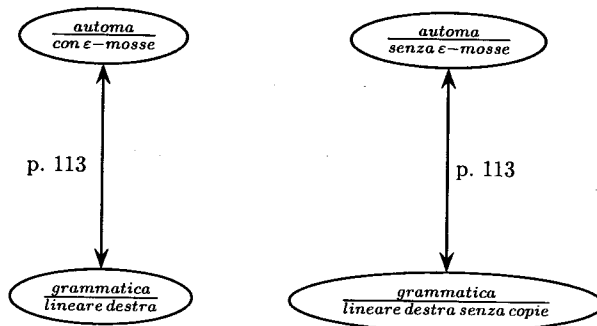
¹²Per la teoria dei linguaggi senza stella si veda McNaughton e Papert [34].



Ad es. si legge nella figura che l'algoritmo GMY converte una e.r. in un automa privo di mosse spontanee.

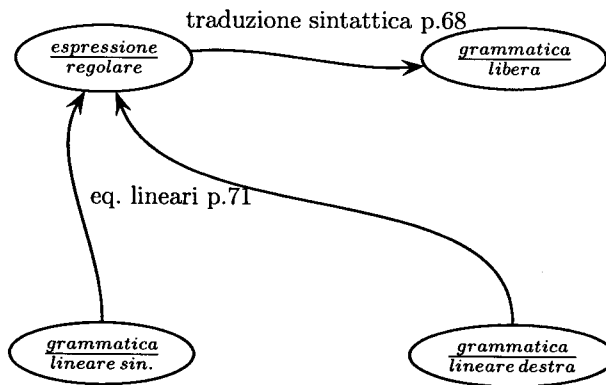
Dove non è indicato il metodo di conversione, la corrispondenza tra i due modelli è immediata.

La prossima figura mostra le corrispondenze dirette tra grammatiche lineari a destra e automi finiti:



Non sono riportate le relazioni tra gli automi e le grammatiche lineari a sinistra, perché sono identiche a quelle con le grammatiche lineari a destra, grazie al principio di dualità sinistra-destra.

Infine si ricordano le relazioni tra espressioni regolari e grammatiche:



Il fatto fondamentale che emerge dal quadro è che le famiglie dei linguaggi regolari, dei linguaggi accettati da automi finiti (deterministici e non) e dei linguaggi generati da grammatiche unilineari (o del tipo 3) coincidono. Infine si ricorda che i linguaggi regolari sono una sottofamiglia assai ristretta di quelli liberi.