



**Università degli Studi di Napoli Federico II**

---

FACOLTÀ DI SCIENZE MM. FF. NN.  
Corso di Laurea Specialistica in Sistemi Informativi

APPUNTI PRIMA LEZIONE DI CALCOLABILITÀ E COMPLESSITÀ

## **Introduzione agli automi su parole infinite**

Candidati:

**Riccardo Tammaro**

Matricola N97/19

**Angelo Russo**

Matricola N97/89

# Indice

<b>1</b>	<b>Introduzione: gli oggetti infiniti</b>	<b>3</b>
<b>2</b>	<b>Automi riconoscitori deterministici e automi produttori deterministici</b>	<b>4</b>
<b>3</b>	<b>Automi riconoscitori</b>	<b>6</b>
3.1	Parole finite . . . . .	7
3.2	Büchi . . . . .	7
3.3	Co-Büchi . . . . .	8
3.4	GB . . . . .	9
3.5	GC . . . . .	9
3.6	Muller . . . . .	9
3.7	Parity . . . . .	9
3.8	Rabin . . . . .	10
3.9	Streett . . . . .	10
<b>4</b>	<b>Automi produttori</b>	<b>11</b>

## Elenco delle figure

1	Automa riconoscitore . . . . .	4
2	Automa produttore . . . . .	5
3	Automa a stati finiti per parole finite ed infinite . . . . .	8
4	Comportamento automa . . . . .	8
5	Automa a stati finiti per parole finite ed infinite . . . . .	12
6	Automa a stati finiti per parole finite ed infinite (Moore) . . .	12
7	Automa a stati finiti per parole finite ed infinite (Mealy) . . .	13

## 1 Introduzione: gli oggetti infiniti

Parlare di oggetti infiniti potrebbe sembrare assurdo dato che tutto ciò che ci circonda é finito. Un calcolatore, ad esempio, é una macchina a stati finiti che può avere un numero particolarmente elevato di stati ma pur sempre finito. Perché allora introdurre questo tipo di oggetti? Consideriamo, a titolo esemplificativo, la progettazione di un sistema software. Tale progettazione potrebbe richiedere, tra le tante cose, un algoritmo di scheduling. Un tale algoritmo ha il compito di allocare i processi ai processori; la principale problematica é legata, qualora i processori non fossero sufficienti in numero ad eseguire tutti i processi parallelamente, alla scelta di quale debba essere il processo, tra quelli in coda, da allocare a quale processore. Teoricamente a questo algoritmo é richiesto di funzionare all'infinito, senza mai terminare. La sua terminazione, infatti, decreterebbe un' attesa permanente dei processi che pertanto non verrebbero mai allocati. Un ulteriore scenario é rappresentato dai server, macchine remote prive di una terminazione prestabilita, che rimangono continuamente in attesa di nuove richieste (a meno che non ci siano problemi tali da forzare un riavvio del server stesso) dei client da eseguire.

## 2 Automi riconoscitori deterministici e automi produttori deterministici

Un automa riconoscitore può essere rappresentato a livello grafico come un box che dato un input ritorna un valore 1 oppure 0 (equivalentemente TRUE o FALSE) a seconda che l'input sia o meno parte del linguaggio su cui l'automata è stato definito. Esempio: consideriamo il linguaggio della logica proposizionale. È uso comune rappresentare le proposizioni atomiche con le lettere maiuscole dell'alfabeto: A, B, ecc.. Le frasi sintatticamente corrette del linguaggio sono dette formule ben formate. Per definizione, l'insieme di formule ben formate è il minimo insieme  $X$  tale che:

- $\perp \in X$ ;
- $A, B, C, \dots \in X$  per ogni simbolo atomico di proposizione;
- Se  $P \in X, \neg P \in X$ ;
- Se  $P, Q \in X, (P \wedge Q), (P \vee Q), (P \rightarrow Q) \in X$ .

Consideriamo, inoltre, un automa definito su tale linguaggio. Il suo compito sarà quello di riconoscere le formule ben formate ritornando in output TRUE (formule corrette in termini sintattici) o FALSE.

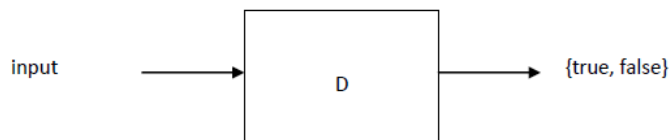


Figura 1: Automa riconoscitore

Un automa produttore, invece, dato un input ritorna una parola su un alfabeto di output, non necessariamente uguale a quello di input. Naturalmente, l'output dipende da come è implementata la macchina al suo interno.

In riferimento ai riconoscitori hanno rilevanza le problematiche decisionali, mentre gli automi produttori trovano un impiego nella formalizzazione dei sistemi, ad esempio, software, hardware, ecc.. La struttura comune ai due tipi di automi è la seguente:

$$D = \langle \Sigma, Q, \delta, q_0 \rangle$$

Dove:

- $\Sigma$  é l'alfabeto di input (puó essere un insieme di lettere oppure di numeri, ecc...);
- $Q$  é l'insieme degli stati;
- $q_0 \in Q$  é lo stato iniziale;
- $\delta : Q \times \Sigma \rightarrow Q$  é la funzione di transizione che, sulla base dello stato attuale e del simbolo letto, indica il prossimo stato in cui la macchina deve transitare.

Considereremo il caso di automi a stati finiti, dunque,  $Q$  sará finito; tuttavia, imporremo una ulteriore restrizione ovvero che anche  $\Sigma$  sia finito. La differenza tra automi riconoscitori e produttori, in termini della struttura  $D$ , consiste in elementi aggiuntivi che verranno trattati successivamente. Si tratta, in breve, di condizioni di accettazione nel caso di automi riconoscitori e di un insieme di output e una funzione di output nel caso di automi produttori. Prima di analizzare la struttura dei due automi definiamo il concetto di input. Per input si intende un oggetto di  $\Sigma^*$  o di  $\Sigma^\omega$ , dove  $\Sigma^*$  e  $\Sigma^\omega$  sono, rispettivamente, l'insieme delle parole finite ed infinite definite su un alfabeto  $\Sigma$ . Formalmente, una parola finita puó essere rappresentata tramite una funzione del tipo:

$$w : [0 \dots n] \rightarrow \Sigma$$

Intuitivamente, dato un indice, la funzione  $w$  ritorna il simbolo  $w(i)$  della parola contenuto in quell'indice. É possibile, tuttavia, utilizzare la notazione alternativa  $w_i$ . Possiamo estendere ora, banalmente, il concetto di parole da finite ad infinite:

$$w : \mathbb{N} \rightarrow \Sigma$$

Sulla base di quanto detto, il linguaggio di un generico automa  $A$ , indicato con  $L(A)$ , é un sottoinsieme di  $\Sigma^*$  o di  $\Sigma^\omega$ , tale per cui una parola finita o infinita  $w$  appartiene a  $L(A)$  sse  $w$  é accettata da  $A$ .

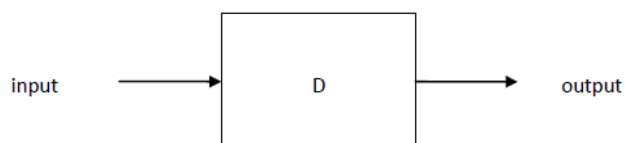


Figura 2: Automa produttore

### 3 Automi riconoscitori

Un riconoscitore deterministico é un automa definito come segue:

$$D = \langle \Sigma, Q, \delta, q_0, \aleph \rangle$$

Il significato degli elementi  $\Sigma, Q, \delta, q_0$  é quello indicato precedentemente.  $\aleph$ , invece, é una condizione di accettazione che, a seconda della sua struttura interna, fa si che si vengano a creare automi con caratteristiche differenti:

1. se  $\aleph = F$  (dove  $F \subseteq Q$ ) avremo allora gli automi individuati rispettivamente con:
  - F: automi a parole finite;
  - B: automi di Büchi;
  - C: automi di co- Büchi;
2. se  $\aleph = \{F_1, \dots, F_k\} \subseteq 2^Q$  gli automi saranno:
  - GB: Generalized Büchi;
  - GC: Generalized co-Büchi;
  - M: Muller;
  - P: Parity. In tal caso, inoltre, vale che  $\{F_1 \subseteq F_2 \subseteq \dots \subseteq F_k = Q\}$ ;
3. se  $\aleph = \{(A_1, B_1), (A_2, B_2), \dots, (A_k, B_k)\} \subseteq 2^Q \times 2^Q$  avremo:
  - R: Rabin;
  - S: Streett.

Prima di definire formalmente la semantica di ognuno di questi automi, occorre introdurre le nozioni di “run di un automa” e “insieme degli stati ripetuti infinitamente spesso”. Tali concetti verranno indicati rispettivamente con *run* e *inf*. Un *run* di un automa deterministico D su una parola  $w$  é una parola finita o infinita sull’alfabeto degli stati che equivale semanticamente al comportamento dell’automa stesso:

$$run_D(w) \in Q^*/Q^\omega$$

Piú specificamente, dire che un automa legge una parola significa che prosegue la sua esecuzione da stato a stato a seconda dell’elemento di input letto. Il *run* di D sulla prima lettera di  $w$  é lo stato iniziale; per induzione, il *run* di D sul simbolo  $i + 1$  di  $w$  é data dalla  $\delta$  sul simbolo  $i$  di  $w$ :

$$run_D(w)_{i+1} = \delta(run(w)_i, w_i)$$

Se la parola  $w$  é finita, detta  $n$  la sua lunghezza, il  $run$  sará una stringa di lunghezza  $n + 1$  e questo per via di uno stato iniziale non dipendente dalla parola e che é comune a tutti i  $run$ . Nel caso di parole infinite, invece, il  $run$  sará semplicemente una parola di lunghezza infinita. Per quanto concerne il concetto di “insieme degli stati ripetuti infinitamente spesso” nel  $run$  vale che:

$$inf(run_D(w)) = \{q \in Q : |\{i \in \mathbb{N} | q = run_D(w)_i\}| = \infty\}$$

Intuitivamente  $inf$  é l’insieme degli stati che si trovano nel  $run$  e che occorrono in un numero di posizioni che é infinito. Passiamo, ora, alla definizione delle condizioni di accettazione a seconda della struttura di  $\aleph$ .

### 3.1 Parole finite

Supposto  $n = |run_D(w)|$ , diremo che  $w \in \Sigma^*$  é accettata da  $D$  sse

$$run_D(w)_{n-1} \in F$$

Ciò significa che l’ultimo stato del  $run$  deve essere di accettazione.

#### Osservazione

Naturalmente l’ $inf$  di un  $run$ , qualunque sia il  $run$ , non é mai vuoto nel caso di oggetti infiniti perché data una parola infinita su un alfabeto finito di stati deve accadere che almeno uno stato si ripeta infinitamente spesso. Se  $Q$  non fosse finito allora ciò non é detto, ovvero non é detto che il  $run$  abbia un oggetto che si ripeta infinitamente spesso. Passiamo, ora, alla definizione di Büchi e co- Büchi.

### 3.2 Büchi

Una parola  $w \in \Sigma^\omega$  é accettata da un automa  $D$  di Büchi sse

$$inf(run_D(w)) \cap F \neq \emptyset$$

Intuitivamente, un  $run$  é di accettazione se e soltanto se l’intersezione tra l’insieme degli stati in cui si viene a trovare l’automata e che occorrono infinitamente spesso e l’insieme  $F$  (insieme degli stati finali) é diversa dal vuoto. Col Büchi, solitamente, si descrivono “proprietá positive” che si desiderano che accadano infinitamente spesso; un esempio é la descrizione del progresso di una computazione.

### 3.3 Co-Büchi

Il co-Büchi, invece, fornisce una descrizione duale ovvero “proprietá negative” che non si vuole che accadono infinitamente spesso. Consideriamo, ad esempio, la progettazione di una macchina che ritorna un codice d’errore a seguito di un input inaspettato; naturalmente, se l’input é corretto, si vuole che la macchina non debba dare errore un numero infinito di volte. Formalmente:

$$\text{inf}(\text{run}_D(w)) \cap F = \emptyset$$

ovvero un *run* é di accettazione se e soltanto se l’intersezione tra l’insieme degli stati in cui si viene a trovare l’automa D e che occorrono infinitamente spesso e l’insieme F (insieme degli stati finali) é uguale al vuoto.

L’esempio che segue é un automa a stati finiti per parole finite ed infinite. Si tratta di un esempio classico, fatto al fine di cogliere la differenza tra il Büchi e il Co-Büchi:

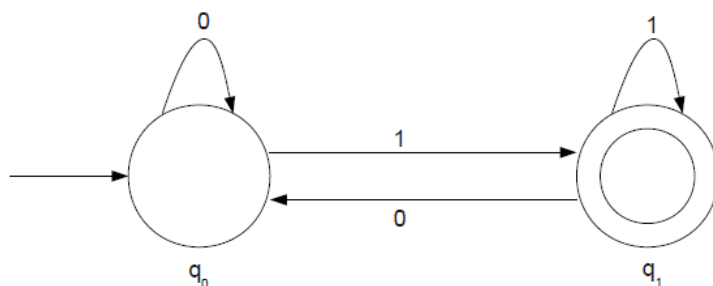


Figura 3: Automa a stati finiti per parole finite ed infinite

Il comportamento dell’automa é definito dalla seguente tabella:

	0	1
q <sub>0</sub>	q <sub>0</sub>	q <sub>1</sub>
q <sub>1</sub>	q <sub>0</sub>	q <sub>1</sub>

Figura 4: Comportamento automa

Se l’automa venisse letto come un automa a parole finite allora il linguaggio da esso accettato sarebbe costituito da tutte quelle stringhe terminanti in

1, ad esempio: 01, 011, 01001, 01111, ecc... Se venisse letto come automa di Büchi allora il linguaggio accettato sarebbe costituito dalle stringhe con una occorrenza infinita di 1. Infine, se venisse letto come un Co-Büchi é necessario che 1 non si presenti infinitamente spesso perché altrimenti l'intersezione con F non sarebbe vuota e pertanto la parola non sarebbe accettata.

### 3.4 GB

Un GB accetta una parola sse

$$\forall i \in [1 \dots k], \text{inf}(\text{run}_D(w)) \cap F_i \neq \emptyset$$

Il che significa che l'automa deve passare infinitamente spesso per almeno uno stato, uno in  $F_1$ , uno in  $F_2$ , uno in  $F_3$ , ecc...

### 3.5 GC

Un GC, invece, accetta una parola sse

$$\exists i \in [1 \dots k] : \text{inf}(\text{run}_D(w)) \cap F_i = \emptyset$$

É possibile notare che le definizioni formali di GB e GC sono una la duale dell'altra.

### 3.6 Muller

Una parola é accettata secondo il criterio di Muller sse

$$\text{inf}(\text{run}_D(w)) \in \mathfrak{K}$$

In questo caso, vengono caratterizzati in modo univoco gli stati che devono essere ripetuti infinitamente spesso e quelli che non devono esserlo, dunque, il *run* deve avere come insieme di stati che si ripetono infinitamente spesso proprio uno tra quelli che vengono esplicitamente detti essere di accettazione. In generale si tratta dell'automa con la piú alta espressivitá in quanto tutti gli altri automi possono essere trasformati in quest'ultimo.

### 3.7 Parity

Per il parity una parola  $w$  é accettata se:

$$\min\{i \in [1 \dots k] : \text{inf}(\text{run}_D(w)) \cap F_i \neq \emptyset\} \text{ é pari.}$$

L'automa di Parity può essere definito, alternativamente, sostituendo  $\min$  con  $\max$ , oppure sostituendo *pari* con *dispari*; aldilà di come viene definito un parity, gli automi risultanti sono tutti equivalenti il che vuol dire che tutto ciò che si riesce a fare con una specifica definizione é possibile farlo con una definizione alternativa.

### 3.8 Rabin

Per Rabin una parola é accettata se  $\exists i \in [1 \dots k]$  tale che:

$$\text{inf}(run_D(w)) \cap \begin{cases} A_i \neq \emptyset \\ B_i = \emptyset \end{cases}$$

La nascita del Rabin é dovuta al fatto che il Büchi e il Co-Büchi, se deterministici, non erano sufficienti ad accettare ciò che era un linguaggio  $\omega$ -regolare; far controllare due condizioni in contemporanea avrebbe portato sicuramente qualche vantaggio. Tuttavia, anche in questo caso, qualche linguaggio non sarebbe stato accettato ed allora se ne sono aggiunte altre.

### 3.9 Streett

Cambiando il simbolo di esistenziale con l'universale e l' $\wedge$  con l' $\vee$ , in breve, si é giunti al duale del Rabin decretando la nascita dello Streett. La presenza dell'universale e dell'or rende lo Streett complesso quanto il Muller a livello di risoluzione.

## 4 Automi produttori

Gli automi produttori vengono definiti come segue:

$$M = \langle \Sigma_I, \Sigma_0, Q, \delta, q_0, \lambda \rangle$$

L'elemento nuovo é  $\lambda$  che é definito diversamente a seconda dell'automa di Moore e di Mealy. L'automa di Moore produce un output in funzione esclusivamente dello stato; ogni stato é etichettato con un simbolo di output, cioé significa che ad una sequenza di stati nella *run* sará associata una sequenza di output. Dunque:

$$\lambda : Q \rightarrow \Sigma_0$$

Per Mealy, invece, l'output non dipende solo dallo stato ma anche dal simbolo in input. La funzione di output é cosí descritta:

$$\lambda : Q \times \Sigma_I \rightarrow \Sigma_0$$

Esempio. Supponiamo di volere una macchina che restituisca un 1 ogni volta che vengono letti due 0 nella parola. Trovati i due 0 la macchina deve restituire un 1, ritornare alla configurazione iniziale ed aspettare i prossimi due 0 (non é detto che i due 0 devono essere consecutivi). Affinché la macchina funzioni correttamente é necessario che essa sia "dotata di memoria". Quando viene letto uno 0 é importante che essa rammenti se ha giá letto un altro 0 oppure no. Nel primo caso dovrá essere restituito in output 1, nel secondo uno 0. É banale, invece, il comportamento che la macchina deve avere quando viene letto un 1 e cioé dare in output uno 0. Bisogna evitare, inoltre, che la macchina, conti una sola volta e cioé che dopo aver trovato due 0 e restituito un 1, ritorni, ogni volta che venga letto uno 0, un 1 in output.

Come realizzare questo comportamento? Supposto di ragionare ancora una volta sull'automa di Fig.3 (riportata di nuovo sotto per comoditá), basterá modificare le etichette degli stati e degli archi come mostrato in Fig. 5

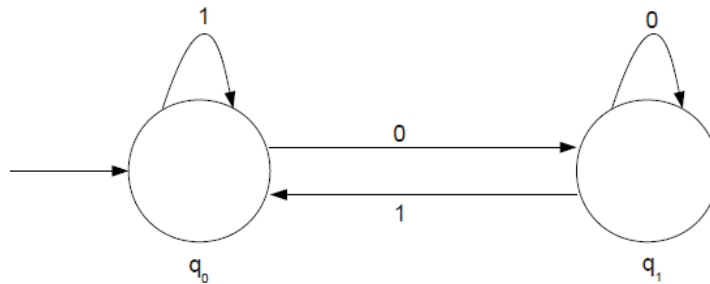


Figura 5: Automa a stati finiti per parole finite ed infinite

Qual é il concetto di output in maniera formalizzata? Il linguaggio di output di un automa di Mealy e di Moore é un sottoinsieme di  $\Sigma_O^*$  o di  $\Sigma_O^\omega$  (ricordiamo, ancora, che  $\Sigma_O^*$  e  $\Sigma_O^\omega$  sono, rispettivamente, l'insieme delle parole finite ed infinite definite su un alfabeto  $\Sigma_O$ ), ovvero:

$$L_{OUT}(M) \subseteq \Sigma_O^*/\Sigma_O^\omega$$

Inoltre, si ha che:

$$L_{OUT}(M) = \{v \in \Sigma_O^*/\Sigma_O^\omega : \exists u \in \Sigma_I^*/\Sigma_I^\omega \text{ per cui } v = out_M(u)\}$$

Nel caso della macchina di Moore si ha che  $v_i = \lambda(run_D(w)_i)$ . Ciò significa che ogni stato in cui la macchina transita viene etichettato sulla base della  $\lambda$  appunto. Consideriamo l'automa di Fig. 6 come una macchina di Moore:

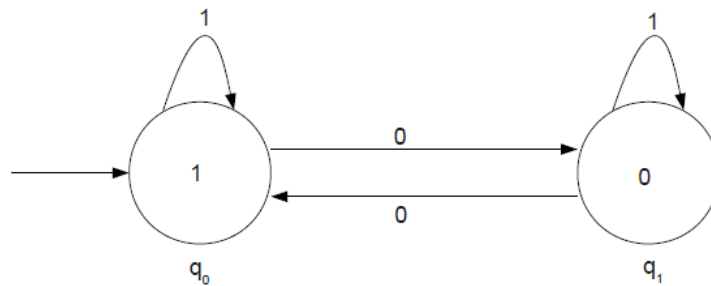


Figura 6: Automa a stati finiti per parole finite ed infinite (Moore)

Dato in input la parola 0 1 1 0, esso si comporta come segue:

- Innanzitutto, viene restituito un 1 iniziale per via dell'etichettatura del primo stato;

- alla lettura del primo 0 ritorna 0;
- alla lettura del primo 1 ritorna 0;
- alla lettura del secondo 1 ritorna 0;
- alla lettura del secondo 0 ritorna 1.

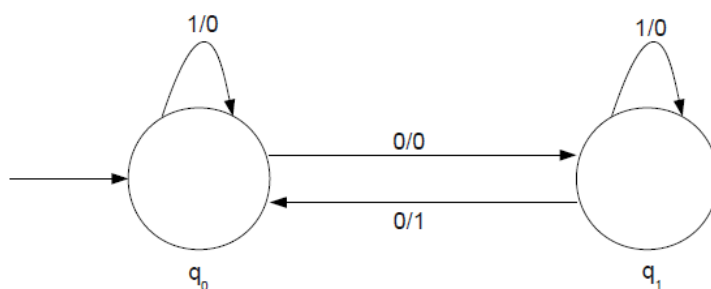


Figura 7: Automa a stati finiti per parole finite ed infinite (Mealy)

L'output dell'automa di Moore ha una lunghezza  $n + 1$ , dove  $n$  é la lunghezza dell'input. Per la macchina di Mealy, invece, la lunghezza dell'output coincide con la lunghezza dell'input e questo perché la  $\lambda$  ha bisogno di un input per produrre qualcosa e quando si é nello stato iniziale non viene prodotto ancora nulla. L'output é definito come segue:

$$v_i = \lambda(\text{run}_D(w)_i, w_i)$$

Quindi, considerando l'automa di Fig. 7, se venisse dato 0110 come input, l'automa di Mealy si comporterebbe come segue:

- alla lettura del primo 0 ritorna 0;
- alla lettura del primo 1 ritorna 0;
- alla lettura del secondo 1 ritorna 0;
- alla lettura del secondo 0 ritorna 1.

Cosa vuol dire che gli automi di Mealy e di Moore sono equivalenti se, come abbiamo visto, l'output ha lunghezza diversa a seconda dell'automa? La diversa lunghezza dipende dal fatto che, nella macchina di Moore anche lo stato iniziale viene labellato. Quindi, a meno della labellatura iniziale l'output ritornato dalle due macchine equivalenti deve coincidere (naturalmente ciò vale per ogni parola del linguaggio dato in input).

Formalmente, detti

$$M_{Moore} = \langle \Sigma_I, \Sigma_O, Q_1, \delta_1, q_{01}, \lambda_1 \rangle$$

$$M_{Mealy} = \langle \Sigma_I, \Sigma_O, Q_1, \delta_2, q_{02}, \lambda_2 \rangle$$

si definisce il concetto di equivalenza nel seguente modo:

$$M_{Moore} \equiv M_{Mealy} \text{ se } \forall w \in \Sigma^* / \Sigma^w \text{ out}_{M_{Mealy}}(w)_i = \text{out}_{M_{Moore}}(w)_{i+1}$$

In particolare, per trasformare un automa di Moore in un automa di Mealy basterá prendere la labellatura di uno stato e fissarla su tutti gli archi uscenti (ovvero verrà dato sempre lo stesso output indipendentemente dal simbolo di input letto successivamente). Per trasformare un automa di Mealy ad un automa di Moore, invece, é necessario labellare lo stato con la label degli archi entranti. In quest'ultimo caso, però, non é detto che gli archi entranti abbiano tutti la stessa label e quindi, al fine di evitare conflitti, é opportuno duplicare lo stato stesso tante volte quanti sono le diverse label degli archi.

Formalmente, per trasformare un automa di Moore ad un automa di Mealy occorre che:

$$Q_2 := Q_1$$

$$\delta_2 := \delta_1$$

$$q_{02} := q_{01}$$

$$\lambda_2(q, \sigma) = \lambda_1(q) \text{ (questo per tutte le } q \text{ e tutte le } \sigma)$$

Per trasformare un automa di Mealy in un automa di Moore, invece, occorre che:

$$Q_1 := Q_2 \times \Sigma_0$$

$$q_{01} := (q_{02}, \sigma_0) \text{ (dove } \sigma_0 \text{ appartiene a } \Sigma_0)$$

$$\delta_1((q, \sigma'), \sigma) = (\delta_2(q, \sigma), \lambda_2(q, \sigma))$$

$$\lambda_1((q, \sigma)) = \sigma$$