Appunti di Informatica Teorica

Riccardo Lo Iacono & Stefano Graffeo

Dipartimento di Matematica & Informatica
Università degli studi di Palermo
Sicilia
a.a. 2022-2023

Indice.

– 1 – Teoria degli automi: introduzione e concetti base.

Si consideri il caso di un interruttore. Grazie agli automi è possibile rappresentare facilmente il passaggio tra i due stati come mostrato in 1.

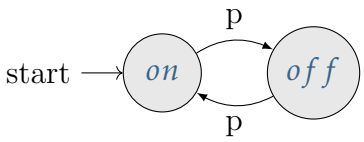


Figura 1: Automa rappresentante uno switch.

Dando una breve definizione di automa: questi è un sistema automatico, rappresentato da un grafo i cui nodi rappresentano gli stati e gli archi le transizioni tra stati.

L'utilizzo degli automi è da ricercare nello studio dei limiti computazionali, cui si legano

1. lo studio della decidibilità, che stabilisce cosa possa fare un computer in assoluto;
2. lo studio della trattabilità che stabilisce cosa possa fare un compute efficientemente.

Agli automi sono inoltre legati due importanti nozioni, quali le grammatiche e le espressioni regolari, che si discuteranno nelle successive sezioni.

– 1.1 – Concetti centrali.

Concetti centrali della teoria degli automi sono gli alfabeti, le stringhe e i linguaggi.

- *Gli alfabeti:* si definisce alfabeto Σ un insieme finito di caratteri.
- *Le Stringhe:* dato Σ un alfabeto, si definisce stringa ω una sequenza di simboli scelti dall'alfabeto.

  Caso particolare è la stringa vuota ε: una stringa composta da zero simboli.

  Data ω una stringa, di questa è possibile stabilirne la lunghezza: ossia il numero di caratteri di cui si compone.

  Infine, considerate $\omega_1 = a_1 \cdots a_k$ e $\omega_2 = b_1 \cdots b_j$ due stringhe, si definisce $\omega_1 \circ \omega_2 = \omega_1 \omega_2 = a_1 \cdots a_k b_1 \cdots b_j$ concatenazione di ω_1 e ω_2.
- *I Linguaggi:* dato Σ un alfabeto, si definisce linguaggio L su Σ un sottoinsieme delle stringe ottenibili con l'alfabeto.

– 2 – Automi DFA e NFA.

Come anticipato in sezione (1): un automa è un sistema automatico, rappresentato da un grafo.
Si tenga presente che esistono due classi di automi

- deterministici o DFA;
- non deterministici o NFA.

– 2.1 – I DFA.

Definizione: Si definisce $A = (Q, \Sigma, \delta, q_0, F)$ DFA, ove

- Q rappresenta l'insieme di stati dell'automa;
- Σ è l'alfabeto utilizzato dall'automa;
- δ definisce le transizioni tra gli stati;
- q_0 indica lo stato iniziale;
- F definisce l'insieme di stati finali;

se considerata δ, per ciascun simbolo dell'alfabeto e per ciascuno stato esiste un'unica transizione per quel carattere.

– 2.1.1 – Funzione di trasizione e funzione di transizione estesa.

Dato un automa A, la funzione di transizione δ stabilisce il comportamento dell'automa in ogni suo stato, per ogni simbolo dell'alfabeto.

Esempio: Sia considerato l'automa di *Figura* (1).
La funzione di transizione dello stesso, definisce le seguenti transizioni

$$\delta(on, p) = (off)$$
$$\delta(off, p) = (on)$$

ossia: letto p dallo stato on passa allo stato off , da questi letto p passa a on.

Definizione: Sia $\omega = a_1 \cdots a_n$ una stringa e δ la funzione di transizione di un dato DFA: si definisce funzione di transizione estesa δ^* la funzione che, letta ω a partire da q_0, stabilisce lo stato di arrivo q_f. Cioè

$$\delta^*(q_0, \omega) = (q_f)$$

Osservazione: Dato un automa, la funzione di transizione estesa δ^*, può essere intesa come la sequenziale applicazione della funzione di transizione δ, per ogni simbolo in ω a partire dallo stato q_0.

– 2.2 – Gli NFA.

Definizione: Si definisce $A = (Q, \Sigma, \delta, q_0, F)$ NFA, ove

- Q rappresenta l'insieme di stati dell'automa;
- Σ è l'alfabeto utilizzato dall'automa;
- δ definisce le transizioni tra gli stati;
- q_0 indica lo stato iniziale;
- F definisce l'insieme di stati finali;

se considerata δ, per ciascun simbolo dell'alfabeto e per almeno uno stato esistono più transizioni per quel carattere.

Esempio: Sia considerato l'automa in *Figura* (1), questi può essere rappresentato come NFA dall'automa in *Figura* (2).

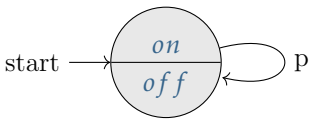


Figura 2: Automa rappresentante uno switch come NFA.

– 2.2.1 – Funzione di transizione estesa.

Definizione: Sia $\omega = a_1 \cdots a_n$ una stringa e δ la funzione di transizione di un dato NFA: si definisce funzione di transizione estesa δ^* la funzione che, letta ω a partire da q_0, stabilisce lo stato di arrivo q_f.

$$\text{Per induzione si ha } \begin{cases} \delta^*(q_0, \varepsilon) = \{q_0\} & \text{base} \\ \delta^*(q_0, \omega) = \bigcup_{q_x \in \delta^*(q_0, \omega)} \delta(q_x, a) \end{cases}$$

– 2.3 – DFA e NFA: linguaggi e proprietà dei linguaggi.

Definizione: Sia A un automa. Si definisce linguaggio di $A, L(A)$, l'insieme delle stringhe ω che accettate da A. Cioè

$$\begin{cases} L(A) = \{\omega : \delta^*(q_0, \omega) \in F\} & \text{se } A \text{ è un DFA} \\ L(A) = \{\omega : \delta^*(q_0, \omega) \cap F \neq \emptyset\} & \text{se } A \text{ è un NFA} \end{cases}$$

– 2.3.1 – Proprietà dei linguaggi.

Sia L il linguaggio riconosciuto da un automa; su di questi è possibile applicare le seguenti operazioni.

- *Potenza n-sima:* si intende la concatenazione di L un certo numero n di volte.

  Esempio: Sia $L = \{\omega : \omega \in \Sigma = \{a, b\}\}$, sia $n = 2$. Segue

  $$L^2 = L \circ L = \{aaaa, aaab, aabb, aaba, abaa, abab, abbb, abba, \dots\}$$

  Osservazione: Se $n = 0$ si ha che $L^0 = \{\varepsilon\}$.

- *Stella di Kleene:* rappresenta l'unione di tutte le potenze di L. Cioè

  $$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$$

  Osservazione: Se $L = \emptyset$ allora $L^* = \{\varepsilon\}$.

- *Croce:* indica l'unione di tutte le potenze di L , meno L^0. Cioè

  $$L^+ = L^1 \cup L^2 \cup \cdots$$

  Vale dunque

  $$L^+ = L \circ L^*$$

– 2.3.2 – Linguaggio universale e complemento.

Definizione: Sia Σ un alfabeto. Si definisce linguaggio universale Σ^*, l'insieme di tutte le parole applicando all'alfabeto Kleene.

Definizione: Sia L un linguaggio su un alfabeto Σ. Si definisce complemento di L, L^C, l'insieme di stringhe che appartengono a Σ^* ma non a L.

– 2.4 – Equivalenza tra NFA e DFA.

Si potrebbe erroneamente pensare che NFA e DFA riconoscano linguaggi diversi, ma si dimostra che non è così.
Prima di dimostrare il teorema di equivalenza tra NFA e DFA, è necessario parlare di *subset construction*.

– 2.4.1 – Subset construction.

Sia $N = (Q = \{q_0, q_1, \ldots, q_k\}, \Sigma, \delta_N, q_0, F_N)$ un NFA.
Per ogni $q_i \in Q, i = 0, 1, \ldots, k$ e per ogni $x \in \Sigma$, si definisco gli stati di un DFA D, dati dall'insieme degli stati definite da δ_N. Inoltre, uno stato di D sara accettante se, almeno uno, degli sti di N da cui è definito è accettante. In fine le transizioni di D, sono analoghe a quelle di N.

Esempio: Sia considerato l'NFA di *Figura* (3)

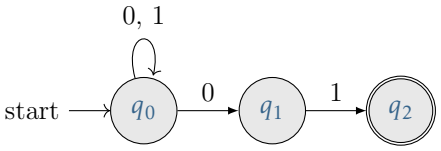


Figura 3: Automa per il linguaggio delle parole che terminano con 01.

Considerando δ si ha

$$\delta(q_0, 0) = (q_0)$$
$$\delta(q_0, 1) = (q_0)$$
$$\delta(q_0, 0) = (q_1)$$
$$\delta(q_1, 1) = (q_2)$$

da ciò segue l'automa di *Figura* (4).

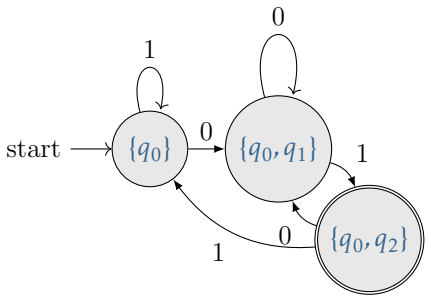


Figura 4: Subset construction dell'automa di *Figura* (3).

Poiché gli stati $\{q_1\}, \{q_2\}$ sono inaccessibili da $\{q_0\}$, questi sono stati trascurati.

– 2.4.2 – Teorema di equivalenza tra NFA e DFA.

Teorema 2.1.
Sia D un DFA ottenuto per subset construction da un NFA N, allora $L(D) = L(N)$.

Dimostrazione: Per dimostrare che $L(D) = L(N)$, si procederà per induzione su $|\omega|$ che

$$\delta_D^*(\{q_0\}, \omega) = \delta_N^*(q_0, \omega) \tag{1}$$

Base: Sia $|\omega| = 0$, ossia $\omega = \varepsilon$.
Per definizione di δ^* , segue che $\delta_D^*(\{q_0\}, \omega) = \delta_N^*(q_0, \omega) = \{q_0\}$.

Induzione: Supposto che quanto detto finora sia vero per $|\omega| = n$, si consideri $|\omega| = n + 1$. Sia posta $\omega = xa$, ove a è l'ultimo carattere della stringa.
Per ipotesi induttiva $\delta_D^*(\{q_0\}, x) = \delta_N^*(q_0, x) = \{p_1, \ldots, p_k\}$, segue dalla definizione induttiva di δ^* per gli NFA

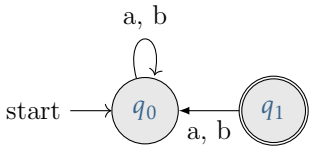
$$\delta_N^*(q_0, \omega) = \bigcup_{i=1}^{k} \delta_N(p_i, a)$$

ma $\bigcup_{i=1}^{k} \delta_N(p_i, a) = \delta_D(\{p_1, \ldots, p_k\}, a)$, segue pertanto

$$\delta_D^*(\{q_0\}, \omega) = \delta_D(\{p_1, \ldots, p_k\}, a) = \bigcup_{i=1}^{k} \delta_N(p_i, a)$$

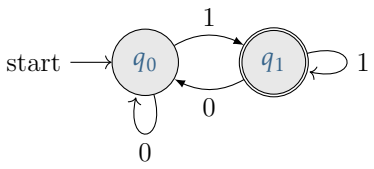
– 2.5 – Esercizi su DFA e NFA.

1. Sia $L = \{\}$ definito su $\Sigma = \{a, b\}$. Si realizzi un automa che lo riconosca.

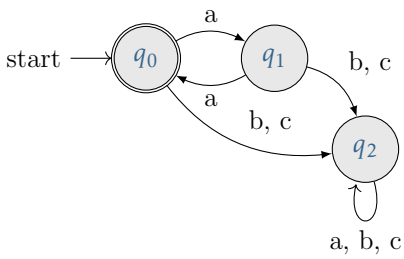


Nota: Soluzione al problema è un qualsiasi DFA, o NFA che sia, al cui stato accettante non è possibile accedere.

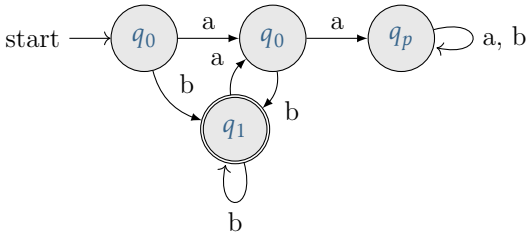
2. Sia $L = \left\{a^{2n} : n \geq 0\right\}$ definito su $\Sigma = \{a, b, c\}$. Si realizzi un automa che lo riconosca.



3. Sia $L = \{\omega : \omega = \Sigma^* 1\}$ definito su $\Sigma = \{0, 1\}$. Si realizzi un automa che lo riconosca.



4. Sia $L = \left\{\omega : \omega = (\Sigma^* aa\Sigma^*)^C\right\}$. Si realizzi un automa che lo riconosca.



– 3 – Proprietà dei linguaggi REC.

Definizione: Sia L un linguaggio. Questi si definisce regolare se accettato da un DFA.

I linguaggi regolari sono chiusi, cioè rimangono regolari, rispetto operazioni quali

- *intersezione*;
- *unione*;
- *complemento*;
- *Kleene*;
- *croce*.

– 3.1 – Chiusura per intersezione.

Teorema 3.1.
Siano L_1 e L_2 linguaggi REC. Allora $L = L_1 \cap L_2$ è REC.

Dimostrazione: Sia A_1 un automa che riconosce L_1 , sia A_2 un automa che riconosce L_2.

$$A_1 = (Q_1, \Sigma, \delta_1, q_{0_1}, F_1) \qquad A_2 = (Q_2, \Sigma, \delta_2, q_{0_2}, F_2)$$

Sia $A = (Q, \Sigma, \delta, q_0, F)$ un automa che riconosce L. Ponendo

- $Q = \{(q_1, q_2) : q_1 \in Q_1 \wedge q_2 \in Q_2\}$ o analogamente $Q = Q_1 \times Q_2$;
- $q_0 = (q_{0_1}, q_{0_2})$;
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ per ogni a tale che la transizione sia definita sia in A_1 che A_2;
- $F = \{(q_1, q_2) : q_1 \in F_1 \wedge q_2 \in F_2\}$ o analogamente $F = F_1 \times F_2$.

Sia $\omega \in L$, segue

$$\begin{aligned} \omega \in L &\iff \omega \in L_1 \wedge \omega \in L_2 \\ &\iff \delta_1^*(q_{0_1}, \omega) \in F_1 \wedge \delta_2^*(q_{0_2}, \omega) \in F_2 \\ &\iff \delta^*((q_{0_1}, q_{0_2}), \omega) \in F \end{aligned}$$

– 3.2 – Chiusura per unione.

Teorema 3.2.
Siano L_1 e L_2 linguaggi REC. Allora $L = L_1 \cup L_2$ è REC.

Dimostrazione: Sia A_1 un automa che riconosce L_1 , sia A_2 un automa che riconosce L_2.

$$A_1 = (Q_1, \Sigma, \delta_1, q_{0_1}, F_1) \qquad A_2 = (Q_2, \Sigma, \delta_2, q_{0_2}, F_2)$$

Sia $A = (Q, \Sigma, \delta, q_0, F)$ un automa che riconosce L. Ponendo

- $Q = \{(q_1, q_2) : q_1 \in Q_1 \wedge q_2 \in Q_2\}$ o analogamente $Q = Q_1 \times Q_2$;
- $q_0 = (q_{0_1}, q_{0_2})$;
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ per ogni a tale che la transizione sia definita sia in A_1 che A_2;
- $F = \{(q_1, q_2) : q_1 \in F_1 \wedge q_2 \in F_2\}$.

Sia $\omega \in L$, segue

$$\begin{aligned} \omega \in L &\iff \omega \in L_1 \vee \omega \in L_2 \\ &\iff \delta_1^*(q_{0_1}, \omega) \in F_1 \vee \delta_2^*(q_{0_2}, \omega) \in F_2 \\ &\iff \delta^*((q_{0_1}, q_{0_2}), \omega) \in F \end{aligned}$$

Nota: Similarmente si dimostra anche la chiusura per complemento, Kleene, croce.

– 4 – Espressioni regolari.

Definizione: Si definisce *espressione regolare*, (o RegEx) e, la descrizione algebrica delle stringhe di un dato linguaggio.

– 4.1 – Costruzione di una RegEx.

Sia e una RegEx. La costruzione di e è di tipo ricorsivo.

Base:

- ε e $\emptyset$ sono espressioni regolari, ove $L(\varepsilon) = \{\varepsilon\}, L(\emptyset) = \{\}$.
- Se α è un simbolo, allora questi è una RegEx, ove $L(\alpha) = \{\alpha\}$.

Induzione:

- Siano e ed f due RegEx. Allora $e + f$ è un'espressione regolare.
- Siano e ed f due RegEx. Allora ef è un'espressione regolare.
- Sia e RegEx. Allora e^* è un'espressione regolare.
- Sia e RegEx. Allora (e) è un'espressione regolare.

– 4.2 – Precedenza nelle RegEx.

Quando si opera con due o più espressioni regolari, bisogna prestare attenzione agli operatori che le lega. In generale, la priorità massima è assegnata alla Stella di Kleene, a seguire la concatenazione e in ultimo la croce.

– 4.3 – Linguaggi locali.

Definizione: Sia L un linguaggio. Questi dicasi locale se esprimibile tramite la seguente quadrupla.

$$(Ini(L), Fin(L), Dig(L), Null(L))$$

ove

- $Ini(L)$ stabilisce l'insieme di caratteri con cui $\omega \in L$ può iniziare.
- $Fin(L)$ stabilisce l'insieme di caratteri con cui $\omega \in L$ può terminare.
- $Dig(L)$ stabilisce l'insieme di tutte le possibili coppie di caratteri che $\omega \in L$ può contenere.
- $Null(L)$ stabilisce se l'insieme contiene o meno la parola vuota.

– 4.3.1 – Calcolo ricorsivo di Ini, Fin, Dig, Null.

Sia L un linguaggio locale. La costruzione della quadrupla, che è ricorsiva, è descritta a seguire.

- **Ini:** considerando la parte ricorsiva
  - $Ini(e+f) = Ini(e) \cup Ini(f)$;
  - $Ini(ef) = Ini(e) \cup Null(e)Ini(f)$;
  - $Ini(e^*) = Ini(e)$.
- **Fin:** considerando la parte ricorsiva
  - $Fin(e+f) = Fin(e) \cup Fin(f)$;
  - $Fin(ef) = Fin(f) \cup Fin(e)Null(f)$;
  - $Fin(e^*) = Fin(e)$.
- **Null:** considerando la parte ricorsiva
  - $Null(e+f) = Null(e) \cup Null(f)$;
  - $Null(ef) = Null(e) \cap Null(f)$;
  - $Null(e^*) = \varepsilon$.
- **Dig:** considerando la parte ricorsiva
  - $Dig(e+f) = Dig(e) \cup Dig(f)$;
  - $Dig(ef) = Dig(e) \cup Dig(f) \cup Fin(e)Ini(f)$;
  - $Dig(e^*) = Dig(e) \cup Fin(e)Ini(e)$.

Nota: Nel descrivere il calcolo di $Ini, Fin, Dig, Null$, è stata trascurata la base. Infatti $*(\varepsilon) = \varepsilon, *(\emptyset) = \emptyset$, ove $*$ sostituisce $Ini, Fin, Dig, Null$, escluso $Dig(\varepsilon) = \emptyset$. Inoltre $Ini(a) = Fin(a) = a$ se a è un carattere, mentre $Dig(a) = Null(a) = \emptyset$.

– 4.3.2 – Automi locali.

Definizione: Sia L un linguaggio locale. Si definisce *automa locale* un DFA che riconosce L.
La costruzione dell'automa locale è realizzata secondo quanto segue.

- $Q = \{q_0\} \cup \Sigma$;
- Se $Null(L) = \varepsilon$ allora q_0 è accettante;
- $\forall a_i \in \Sigma$, ogni arco etichettato a_i, entra nello stato q_{a_i};
- Da q_0 escono le transizioni definite da Ini;
- Le altre transizioni sono definite da Dig;
- Gli stati finali sono indicati da Fin.

– 4.4 – RegEx e automi locali.

Definizione: Sia e una RegEx. Questa si dice lineare se nessun carattere in e è ripetuto.

Sia e una RegEx non lineare. Questa può essere linearizzata semplicemente ridefinendo le occorrenze multiple, così che l'espressione e' ottenuta dalla linearizzazione sia definito su un nuovo alfabeto Σ'.

Algoritmo .
Sia e espressione regolare. La costruzione di un'automa locale che riconosce e è realizzata come segue.

1. *Se e è regolare si al punto 2, altrimenti si procede alla linearizzazione.*
2. *Si definisce da quadrupla $(Ini(L), Fin(L), Dig(L), Null(L))$, procedendo ricorsivamente al calcolo degli insiemi.*
3. *Si costruisce l'automa locale seguendo le transizioni della quadrupla.*
4. *Se la quadrupla è definita dopo aver linearizzato e, allora si procede rimuovendo la ridefinizione dei caratteri.*
5. *Si procede alla subset construction.*

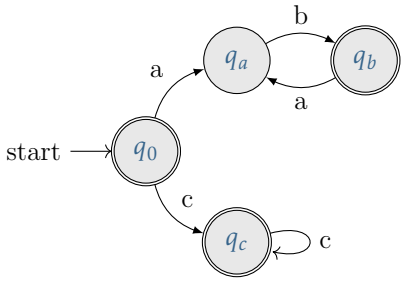
Esempio: Sia $e = (ab)^* + c^*$. Si costruisce un automa che riconosca e.

Svolgimento: Procedendo applicando l'algoritmo si ha quanto segue.

1. Si osserva che e è lineare, si passa dunque alla costruzione della quadrupla, da cui

   - $Ini(e) = \{a, c\}$
   - $Fin(e) = \{b, c\}$
   - $Dig(e) = \{ab, ba, cc\}$
   - $Null(e) = \varepsilon$

2. Procedendo alla costruzione dell'automa, segue



Osservazione: Poiché e è lineare si ha che non è necessario procedere alla subset construction. Infatti l'automa di cui sopra è gia un DFA.

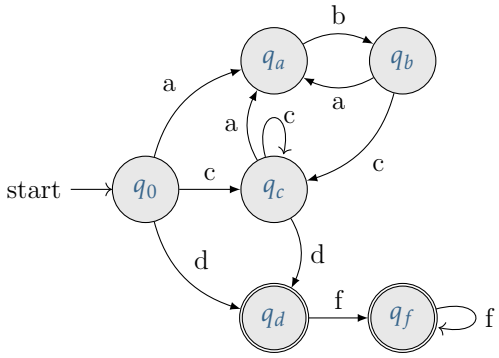
Esempio: Sia $e = (ab + a)^* ba^*$. Si costruisca un automa che riconosca e.

Svolgimento: Procedendo applicando l'algoritmo si ha quanto segue.

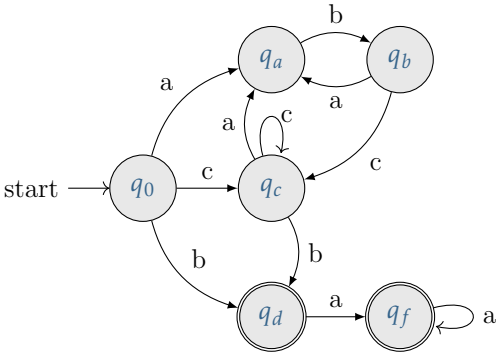
1. Si osserva che e non è lineare, si procede alla sua linearizzazione. Segue che
$$e = (ab + a)^* ba^* \quad \text{diventa} \quad e' = (ab + c)^* df^*$$

2. Considerando la quadrupla, segue
   - $Ini(e') = \{a, c, d\}$
   - $Fin(e') = \{d, f\}$
   - $Dig(e') = \{ab, bc, ca, ba, \ldots\}$
   - $Null(e') = \emptyset$

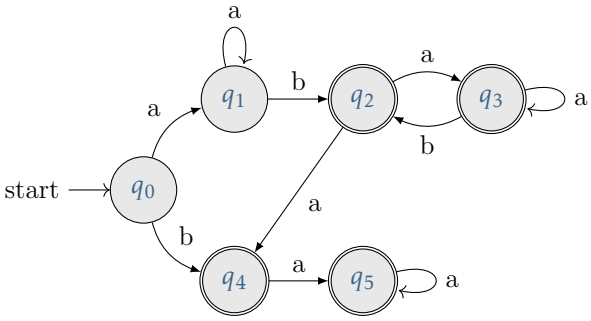
3. Passando all'automa, segue



4. Procedendo rimuovendo la ridefinizione dei caratteri, si ha



5. Concludendo con la subset construction, segue



– 4.5 – Da DFA a RegEx.

Si è finora dimostrato che per ogni RegEx esiste un automa che lo riconosce. Si può dimostrare il viceversa, se il seguente teorema è soddisfatto.

Teorema di Kleene.
Sia L un linguaggio regolare, sia A un DFA che lo riconosce. Allora esiste un'espressione regolare e equivalente.

Più in generale

$$REC = REG$$

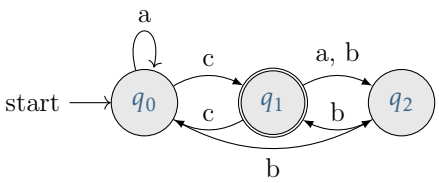
ove REG indica l'insieme dei linguaggi regolari.

Algoritmo .
Sia A un DFA che riconosce un certo linguaggio L. La costruzione della RegEx equivalente è realizzata come segue.

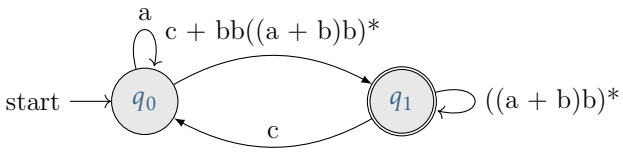
- *Se A ha più stati accettanti, si creano tante copie quante gli stati finali, ciascuno con un solo stato accettante.*
- *Per ciascuna delle copie:*
  - *si eliminano le transizioni intermedie, fino ad ottenere automi con un solo stato accettante e uno finale;*
  - *si determina la RegEx e_i per la copia.*
- *L'espressione regolare sarà data come*

$$e = e_1 + \cdots + e_k$$

Esempio: Sia A il seguente DFA. Si trovi la RegEx equivalente.



Svolgimento: Poiché q_2 è l'unico stato che non è ne finale ne iniziale, si procede alla sua eliminazione. Da cio segue quanto nella figura a seguito.



Da cui l'espressione equivalente è

$$e = a^*(c + bb((a+b)b)^*)(c(c + bb((a+b)b)^*) + ((a+b)b)^*)$$

– 5 – Proprietà dei linguaggi regolari.

Siano A , B due DFA. Si può dimostrare che

$$A = B \iff L(A) = L(B)$$

– 5.1 – Minimizzazione di DFA.

Sia A un DFA. A seguito di quanto detto sopra, ne consegue che è possibile realizzare un DFA B equivalente ad A, ma con un numero minimo di stati. Analogo ragionamento è estensibile agli NFA, sebbene per questi non è sempre vero.

– 5.1.1 – Relazione di indistinguibilità.

Sia A un DFA, siano p e q suoi stati. Si ha che

$$p \,|\, q \iff (\delta^*(q,\omega) \wedge \delta^*(q,\omega)) \in F, \quad \forall \omega \in \Sigma^*$$

oppure

$$p \,|\, q \iff (\delta^*(q,\omega) \wedge \delta^*(q,\omega)) \notin F, \quad \forall \omega \in \Sigma^*$$

Cioè p e q sono indistinguibili se per ogni parola del linguaggio universale si ha che, calcolando la funzione di transizione estesa per i due stati, entrambi conducono ad uno stato accettante/rifiutante per ω.

– 5.1.2 – Algoritmo riempi-tabella.

Uno strumento utile alla minimizzazione è l'algoritmo riempi-tabella, con il quale è possibile stabilire ricorsivamente gli stati equivalenti.

Base: Se p è accettante e q non lo è, allora la coppia (p,q) è distinguibile.

Induzione: Se p,q sono stati tali che, per un simbolo di input α, si ha che

$$\delta(p,\alpha) \wedge \delta(q,\alpha)$$

conducono a stati noti come distinguibili, allora (p,q) sono distinguibili.

Teorema 5.1.
Se due stati non sono distinti dall'algoritmo riempi-tabella, alloro sono equivalenti.

– 5.2 – Pumping Lemma.

Capita spesso di perdere molto tempo nello stabilire se un linguaggio L è regolare o meno. Per semplificare tale processo è possibile utilizzare uno strumento molto potente: il *pumping lemma*.

Lemma 5.2.1.
Sia L un linguaggio. Questi non è regolare se

$$\forall n : \exists \omega \in L, |\omega| \leq n, \exists x, y, z : \omega = xyz$$

per cui almeno una delle seguenti proprietà non è soddisfatta.

- $y \neq \varepsilon$
- $|xy| \leq n$
- $\forall k \geq 0, xy^k z \in L$

Esercizio: Sia $L = \{\omega \in \Sigma = \{a, b\} : \omega = a^n b^n\}$. Stabilire se L è regolare.

Svolgimento: Procedendo applicando il pumping lemma

$$\forall n, \qquad a^n b^n \in L$$

cioè

$$\underbrace{a \cdots a}_{n \text{ volte}} \underbrace{b \cdots b}_{n \text{ volte}} \in L$$

Considerando ora le partizioni valide

1. $x = \underbrace{a \cdots a}_{i \text{ volte}}, y = \underbrace{a \cdots a}_{j \text{ volte}}, z = \underbrace{b \cdots b}_{n \text{ volte}}$ con $i + j = n$.
2. $x = \underbrace{a \cdots a}_{i \text{ volte}}, y = \underbrace{a \cdots a}_{j \text{ volte}} \underbrace{b \cdots b}_{l \text{ volte}}, z = \underbrace{b \cdots b}_{h \text{ volte}}$ con $i + j = n, l + h = n$.

si ha che la prima viola la terza proprietà per $k = 0$, la seconda partizione, e in generale quelle che richiedono almeno una b, violano la seconda proprietà.