Appunti di Informatica Teorica

Riccardo Lo Iacono

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.

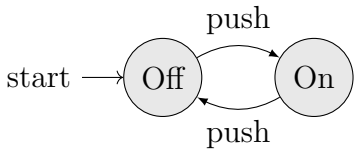


Figura 1: Automa rappresentante un interruttore.

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 trattabilita, che stabilisce cosa possa fare un compute efficientemente.

Agli automi sono inoltre legati due importanti nozioni, quali

- le grammatiche: modelli utili alla progettazione di software che elaborano dati con struttura ricorsiva;
- le espressioni regolari: denotano la struttura dei dati, specie se i dati sono stringhe.

– 1.1 – Concetti centrali.

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

– 1.1.1 – Gli alfabeti.

Un *alfabeto* è un'insieme finito e non vuoto di simboli, convenzionalmente indicato con Σ.

– 1.1.2 – Le stringhe.

Una *stringa* è una sequenza di simboli scelti da un alfabeto. Caso particolare di stringa è la stringa ϵ (o stringa vuota): cioè composta da zero simboli.

Di una stringa è possibile inoltre stabilire la lunghezza, la quale non è altro che il numero di simboli di cui la stessa si compone.
Generalmente si indica la lunghezza di una stringa ω come $|\omega|$.

Le stringhe, posso anche essere concatenate: cioè, supponendo le stringhe $x = a_1 a_2 \dots a_i$ e $y = b_1 b_2 \dots b_j$, la stringa $x \circ y$ (o xy) $= a_1 a_2 \dots a_i b_1 b_2 \dots b_j$.

– 1.1.3 – I linguaggi.

Considerato un alfabeto Σ, un linguaggio L è un sottoinsieme di tutte le stringhe componibili con Σ.

– 2 – Automi e DFS.

Come anticipato in Sezione 1: un automa è un sistema automatico rappresentato da un grafo. Questi sono classificabili in due categorie

1. automi non deterministici;
2. automi deterministici.

Definizione.
Si definisce automa deterministico (DFA), la quintupla $(Q, \Sigma, \delta, q_0, F)$, ove

- Q è un insieme finito di stati;
- Σ è un insieme finito di simboli in input;
- δ è una funzione, detta funzione transitoria;
- q_0 è lo stato iniziale dell'automa;
- F è l'insieme di tutti gli stati accettanti.

– 2.1 – Funzione transitoria e funzione transitoria estesa.

La funzione transitoria stabilisce il comportamento dell'automa in ogni suo stato, per ogni elemento di Σ.

Considerando Figura 1, la funzione transitoria associata può essere rappresentata come segue. A partire da una funzione transitoria,

-	push
off	on
on	off

Tabella 1: Rappresentazione tabulare della funzione transitoria di Figura 1

è possibile definire la funzione transitoria estesa $\overline{\delta}$. Questa riceve i parametri q_0, w e restituisce s, ove

- q_0 è lo stato iniziale dell'automa;
- w una stringa di caratteri appartenenti a Σ;
- s lo stato di arrivo.

Osservazione.
La funzione transitoria estesa può essere intesa come una sequenziale applicazione di δ a partire da q_0 per ogni carattere in w.

– 2.2 – Linguaggio di una DFA e proprietà dei linguaggi.

Dato un automa deterministico A, si definisce linguaggio $L(A)$, l'insieme delle stringhe w che partendo da uno stato iniziale portano a uno finale. Cioè

$$L(A) = \left\{ w \,\middle|\, \overline{\delta}(q_0, w) \in F \right\}$$

In generale ad un linguaggio è possibili applicare delle proprietà, quali

- *potenza n-sima*: consiste nella concatenazione di n volte L.

  Esempio.
  Sia $L = \{a, b\}$, la potenza n-sima di L, con n = 2, risulta essere quanto segue.

  $$L^2 = L \circ L = \{aa, ab, ba, bb\}$$

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

- *stella di Kleene*: consiste nell'unione di tutte le potenze di un linguaggio. Cioè

  $$L^* = \bigcup_{0 \le i \le n} L^i = L^0 \cup L^1 \cup \cdots \cup L^n$$

- *croce*: equivalente alla stella di Kleene, con la differenza che è esclusa la parola vuota. Cioè

  $$L^+ = \bigcup_{1 \le i \le n} L^i = L^1 \cup L^2 \cup \cdots \cup L^n$$

  questi può essere visto come segue.

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

– 2.3 – Linguaggio universale e complemento.

Sia Σ un alfabeto, si definisce Σ^* linguaggio universale l'insieme di tutte le parole ottenute da Σ applicando ad esso la stella di Kleene. Quindi se $\Sigma = \{\varepsilon, a_1, a_2\}$, segue che

$$\Sigma^* = \{\varepsilon, a_1, a_2, a_1a_2, a_2a_1, \dots\}$$

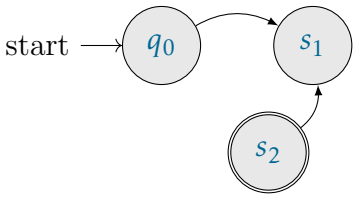
Considerando un linguaggio L su un alfabeto, si definisce $\neg L$ complemento di L: cioè l'insieme delle parole appartenenti a Σ^* ma non ad L. cioè

$$\neg L = \Sigma^* - L$$

– 3 – Esercizi di riepilogo.

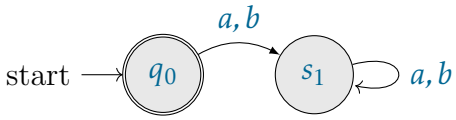
Di seguito si darà la soluzione ad alcuni esercizi, utili alla comprensione degli argomenti finora trattati.

1. Si realizzi un DFA che gestisca un linguaggio vuoto: L = {}.

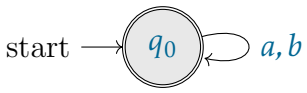


Osservazione.
Soluzione al problema risulta essere un qualsiasi DFA il cui stato accettante risulta inaccessibile.

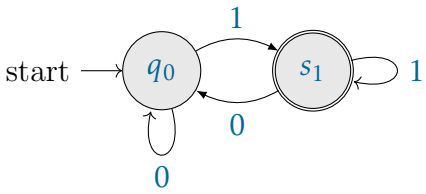
2. Si realizzi un automa che dato $\Sigma = \{a, b\}$, accetti come linguaggio $L = \{\varepsilon\}$.



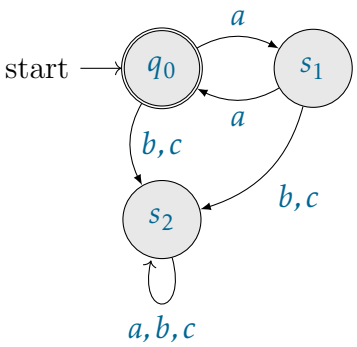
3. Si realizzi un automa che gestisca Σ^*, con $\Sigma = \{a, b\}$.



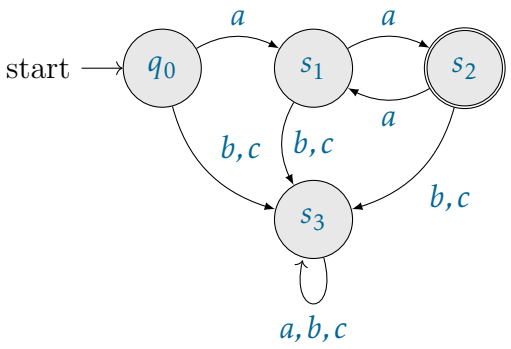
4. Si realizzi un automa che gestisca le stringhe binarie che terminano con 1.



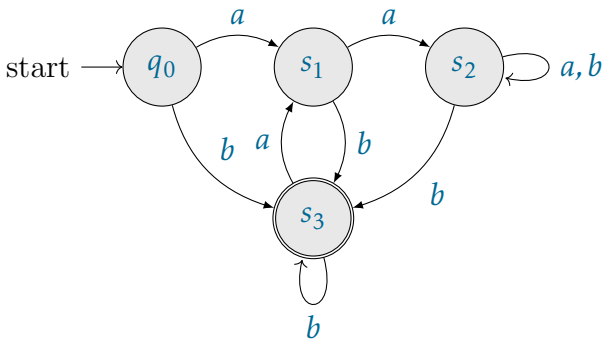
5. Si realizzi un automa che, dato $\Sigma = \{a, b, c\}$, accetti unicamente parole con un numero pari di 'a', compresa la parola vuota.
   ATTENZIONE: una parola del tipo $w = aabc$, deve essere rifiutata.



6. Si realizzi un automa che, dato $\Sigma = \{a, b, c\}$, accetti unicamente parole con un numero pari di 'a', esclusa la parola vuota.
   ATTENZIONE: una parola del tipo $w = aabc$, deve essere rifiutata.



7. Si realizzi un automa che, dato $\Sigma = \{a, b\}$, gestisca parole terminanti con 'b', che non presentano la sequenza 'aa'.



– 4 – Automi non deterministici.

Definizione.
Si definisce automa non deterministico (NFA), la quintupla $(Q, \Sigma, \delta, q_0, F)$. Come facilmente intuibile, un NFA ha le stesse componenti di un DFA; questi però differisce dal ruolo svolto da δ, la quale non restituirà più un solo stato finale bensì un insieme di questi.

– 4.1 – Funzione estesa di una NFA.

Come nel caso di una DFA, anche per le NFA è possibile stabilire una funzione estesa $\overline{\delta}$. Questa anziché operare su un unico simbolo, opererà su una stringa.

In termini formali:
$$\begin{cases} \overline{\delta}(q_0, \varepsilon) = \{q_0\} & \text{base} \\ \overline{\delta}(q_0, \omega a) = \bigcup_{p \in \overline{\delta}(q_0, \omega)} \delta(p, a) \end{cases}$$

– 4.2 – Linguaggio di un NFA.

Sia ω una stringa di caratteri, si dirà che l'NFA A accetta ω se esiste almeno un percorso che da uno stato iniziale q_0, conduce ad uno stato di accettazione. Cioè

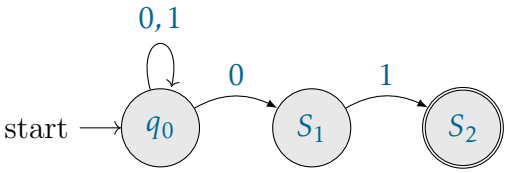
$$L(A) = \left\{ \omega \in \Sigma^* \,\middle|\, \overline{\delta}(q_0, \omega) \cap F \neq \varnothing \right\}$$

– 4.3 – Equivalenza tra NFA e DFA.

Esistono linguaggi per cui risulta più semplice realizzare un'NFA che un DFA. Per quanto sorprendente però, NFA e DFA sono equivalenti in quanto, come si dimostrerà in seguito, questi, oltre che accettare gli stessi linguaggi, sono reciprocamente convertibili.

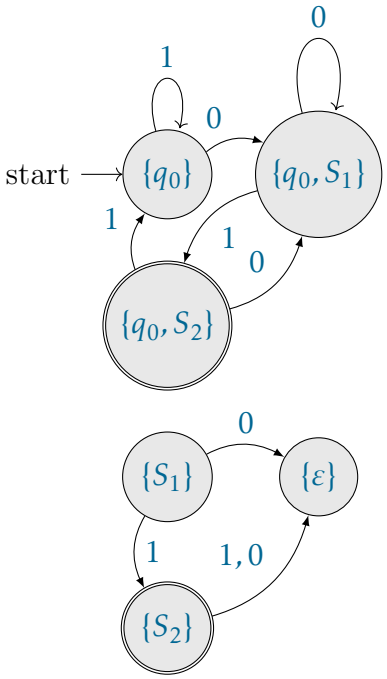
Prima di dimostrare che $L_{NFA} = L_{DFA}$ è necessario parlare di *costruzione per sottoinsiemi*. Questa stabilisce che: dato un NFA $A = (Q_N, \Sigma, \delta_N, q_0, F_N)$ è possibile creare un DFA $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ equivalente.

Si consideri il seguente NFA

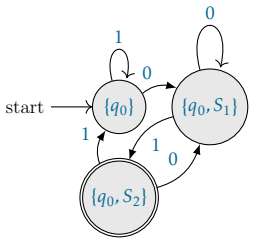


si dimostrerà dapprima tramite esempio, successivamente formalmente che questi è convertibile in un DFA equivalente.

Si parta col considerare tutti i sottoinsiemi ottenibili tramite i percorsi dell'NFA. Si ha che



Si nota però che gli stati $\{S_1\}, \{S_2\}, \{\varepsilon\}$, risultano inaccessibili dallo stato iniziale, pertanto è possibile trascurarli. Da ciò segue pertanto che il DFA equivalente è



Dando adesso una dimostrazione formale, si enuncerà e dimostrerà il teorema per la costruzione tramite sottoinsiemi.

Teorema Rabin-Scott.
Se $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ è un DFA ottenuto da un NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$, mediante costruzione per sottoinsiemi, allora $L(D) = L(N)$.

Dimostrazione.
Si dimostra per induzione che

$$\overline{\delta}_D(\{q_0\}, \omega) = \overline{\delta}_N(q_0, \omega) \tag{1}$$

Base.
Sia $|\omega| = 0$, cioè $\omega = \varepsilon$, per la definizione di $\overline{\delta}$, si ha che $\overline{\delta_D}(\{q_0\}, \omega) = \overline{\delta_D}(q_0, \omega) = \{q_0\}$.

Induzione.
Supposto vero l'enunciato per $|\omega| = n$, si considera $|\omega| = n + 1$; scomponendo ω in $\omega = x\alpha$ ove α è l'ultimo simbolo di ω.
Per ipotesi induttiva $\overline{\delta_D}(\{q_0\}, x) = \overline{\delta_D}(q_0, x)$, siano gli stati di questi insiemi $\{p_1, \ldots, p_k\}$.

Dalla definizione induttiva di $\overline{\delta}$ per gli NFA si ha

$$\overline{\delta}_N(q_0, \omega) = \bigcup_{i=1}^{k} \delta_N(p_i), a \tag{2}$$

ma si ha che

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

da ciò segue pertanto

$$\overline{\delta}_D(q_0, \omega) = \delta_D(\{p_1, \ldots, p_k\}, a) = \bigcup_{i=1}^{k} \delta_N(p_i), a \tag{3}$$

Confrontando le Equazioni (2), (3) si dimostra l'Equazione (1).

– 5 – Proprietà di chiusura di REC.

Si definisce REC la famiglia di linguaggi accettati da un automa a stati finiti. Questi è chiuso rispetto le operazioni di

- intersezione;
- unione;
- complemento.

Osservazione.
Similarmente si dimostra la chiusura per le altre operazioni.

– 5.1 – Intersezione.

Teorema 5.1.
Siano L_1 e L_2 linguaggi REC, allora il linguaggio $L_1 \cap L_2$ appartiene a REC.

Dimostrazione.
Siano A_1, A_2 automi tali che A_1 riconosca L_1 e A_2 riconosca L_2, con

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

L'automa A risultante dalla loro intersezione sarà uguale alla quintupla $(Q, \Sigma, \delta, q_0, F)$, ove

- $Q = \{(q_1, q_2) \mid q_1 \in Q_1 \wedge q_2 \in Q_2\}$ oppure $Q = Q_1 \times Q_2$;
- $q_0 = (q_{0_1}, q_{0_2})$, ossia la coppia di stati iniziali;
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ per ogni a tale che la transizione sia definita per ambedue gli automi.
- $F = \{(q_1, q_2) \mid q_1 \in F_1 \wedge q_2 \in F_2\}$ oppure $F = F_1 \times F_2$;

Sia pertanto $\omega \in L_1 \cap L_2$, si dimostra che A la riconosce.

$$\begin{aligned}
\omega \in L_1 \cap L_2 &\iff \omega \in L_1 \wedge \omega \in L_2 \\
&\iff \overline{\delta}_1(q_{0_1}, \omega) \in F_1 \wedge \overline{\delta}_2(q_{0_2}, \omega) \in F_2 \\
&\iff (\overline{\delta}_1(q_{0_1}, \omega), \overline{\delta}_2(q_{0_2}, \omega)) \in F \\
&\iff \overline{\delta}((q_{0_1}, q_{0_2}), \omega) \in F
\end{aligned}$$

– 5.2 – Unione.

Teorema 5.2.
Siano L_1 e L_2 linguaggi REC, allora il linguaggio $L_1 \cup L_2$ appartiene a REC.

Dimostrazione.
Siano A_1, A_2 automi tali che A_1 riconosca L_1 e A_2 riconosca L_2, con

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

L'automa A risultante dalla loro unione sarà uguale alla quintupla $(Q, \Sigma, \delta, q_0, F)$, ove

- $Q = \{(q_1, q_2) \mid q_1 \in Q_1 \vee q_2 \in Q_2\}$ oppure $Q = Q_1 \times Q_2$;
- $q_0 = (q_{0_1}, q_{0_2})$, ossia la coppia di stati iniziali;
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ per ogni a tale che la transizione sia definita per ambedue gli automi.
- $F = \{(q_1, q_2) \mid q_1 \in F_1 \vee q_2 \in F_2\}$ oppure $F = F_1 \times Q_2 \wedge Q_1 \times F_2$;

Sia pertanto $\omega \in L_1 \cup L_3$, si dimostra che A la riconosce.

$$\begin{aligned} A \text{ accetta } \omega &\iff \overline{\delta}((q_{0_1}, q_{0_2}), \omega) \in F \\ &\iff (\overline{\delta}_1(q_{0_1}, \omega), \overline{\delta}_2(q_{0_2}, \omega)) \in F \\ &\iff \overline{\delta}_1(q_{0_1}, \omega) \in F_1 \vee \overline{\delta}_2(q_{0_2}, \omega) \in F_2 \\ &\iff \omega \in L_1 \vee \omega \in L_2 \end{aligned}$$

ma ciò implica $\omega \in L_1 \cup L_2$.

– 6 – Espressioni regolari (Regex).

Le *espressioni regolari* forniscono una descrizione algebrica dei linguaggi, fornendo inoltre un modo dichiarativo per esprimere quali stringhe siano accettate.

– 6.1 – Costruzione di una Regex.

Come altre algebre, anche l'algebra delle Regex presenta variabili e costanti.

In genere un'espressione regolare è espressa ricorsivamente come segue

Base.

- ε e $\varnothing$ sono Regex, le quali rappresentano i linguaggi $L(\varepsilon) = \{\varepsilon\}, L(\varnothing) = \varnothing$;
- Se α è un simbolo qualsiasi, allora α è una Regex ed equivale a $L(\alpha) = \{\alpha\}$.

Induzione.

- Siano E ed F espressioni regolari, allora $E + F$ è espressione regolare.
- Siano E ed F espressioni regolari, allora $E \circ F$ è espressione regolare.
- Siano E espressione regolare, allora E^* è espressione regolare.
- Se E è una Regex, allora (E) è una Regex.

– 6.2 – Precedenza degli operatori delle Regex.

Anche le Regex sono soggette a regole relative l'ordine degli operatori.

In breve: $*$ ha precedenza su $\circ$ che ha precedenza su $+$.

– 6.3 – Linguaggi locali.

Dato L un linguaggio, questi si definisce locale se lo stesso può essere espresso dalla seguente quadrupla

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

ove

- $Ini(L) = \{a \in \Sigma \mid a\Sigma^* \cap L \neq 0\}$
- $Fin(L) = \{a \in \Sigma \mid \Sigma^* a \cap L \neq 0\}$
- $Dig(L) = \left\{u \in \Sigma^2 \,\middle|\, \Sigma u \Sigma \cap L \neq 0\right\}$
- $Null(L)$: se $\{\varepsilon\} \in L$ o meno: se lo è $Null(L) = \varepsilon$, altrimenti $Null(L) = \varnothing$.

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

Si descrive di seguito un metodo per il calcolo di Ini, Fin, Dig, Null di un'espressione regolare.

– 6.4.1 – Ini.

Considerando i casi base

- $Ini(\varepsilon) = \varepsilon$;
- $Ini(\varnothing) = \varnothing$.

Procedendo: dati e ed e_1 espressioni regolari, si ha

- $Ini(e + e_1) = Ini(e) \cup Ini(e_1)$;
- $Ini(ee_1) = Ini(e \circ e_1) = Ini(e) \cup Null(e) Ini(e_1)$;
- $Ini(e^*) = Ini(e)$.

– 6.4.2 – Fin.

Considerando i casi base

- $Fin(\varepsilon) = \varepsilon$;
- $Fin(\varnothing) = \varnothing$.

Procedendo: dati e ed e_1 espressioni regolari, si ha

- $Fin(e + e_1) = Fin(e) \cup Fin(e_1)$;
- $Fin(ee_1) = Fin(e \circ e_1) = Fin(e_1) \cup Null(e) Fin(e)$;
- $Fin(e^*) = Fin(e)$.

– 6.4.3 – Null.

Considerando i casi base

- $Null(\varepsilon) = \varepsilon$;
- $Null(\varnothing) = \varnothing$.

Procedendo: dati e ed e_1 espressioni regolari, si ha

- $Null(e + e_1) = Null(e) \cup Null(e_1)$;
- $Null(ee_1) = Null(e \circ e_1) = Null(e) \cap Null(e)$;
- $Null(e^*) = \varepsilon$.

– 6.4.4 – Dig.

Considerando i casi base

- $Dig(\varepsilon) = \varnothing$;
- $Dig(\varnothing) = \varnothing$.

Procedendo: dati e ed e_1 espressioni regolari, si ha

- $Dig(e + e_1) = Dig(e) \cup Dig(e_1)$;
- $Dig(ee_1) = Dig(e \circ e) \cup Dig(e_1) \cup Fin(e)Ini(e_1)$;
- $Dig(e^*) = Dig(e) \cup Fin(e)Ini(e)$.

– 6.5 – Esercizio esemplificativo Regex.

Sia $e = a(bc^* + d)^* f^*$ espressione regolare, si stabilisca se e è locale.

- $Ini(e)$: sebbene sia ovvio che Ini(e) = a, si verificherà quanto affermato osservando l'albero che compone e.

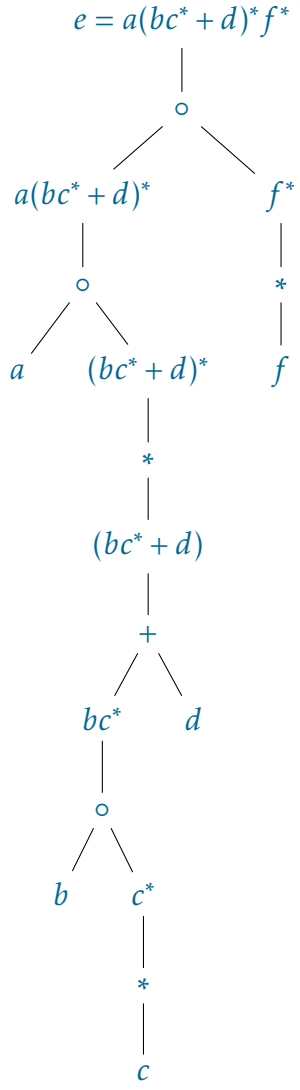


Figura 2: Schema della Regex e = a(bc* + d)*f*.

Applicando quanto detto per il calcolo ricorsivo di Ini(e) = a.

- $Fin(e)$: considerando Figura (2), si nota che le proprietà per il calcolo ricorsivo di Fin, $Fin(e) = \{a, b, c, d, f\}$.
- $Dig(e)$: considerando di nuovo Figura (2), da questa risulta evidente che $Dig(e) = \{ab, ad, af, bb, bc, bd, bf, cb, cc, cd, cf, db, dd, df, ff\}$.
- $Null(e)$: considerando Figura (2), si nota che $Null(e) = \varnothing$.

– 6.6 – Automi locali.

Definizione.
Si definisce *locale* un DFA A, non necessariamente completo, che riconosce un linguaggio locale L.

Questi è definito come segue.

- $Q = \{q_0\} \cup \Sigma$;
- se $Null(L) = \varepsilon \implies q_0$ accettate;
- $\forall a_i \in \Sigma$, ogni arco con etichetta a_i, entra in S_{a_i};
- gli stati finali sono indicati da $Fin(L)$
- da q_0 escono tutte le transizioni definite in $Ini(L)$;
- le transizioni tra i vari nodi sono definiti da $Dig(L)$.

Osservazione.
Una transizione è definite da q_a a q_b se $'ab' \in Dig(L)$.

– 6.7 – Costruzione di DFA data una Regex.

Prima di descrivere l'algoritmo per la costruzione di un DFA a partire da una Regex, è necessario dare la definizione di espressione lineare.

Definizione.
Data un'espressione regolare e, questa si dice *lineare* se nessun carattere in e si ripete.

Se un'espressione non è lineare, questa può essere linearizzata rinominando le occorrenze multiple, tale che l'espressione e' ottenuta sia definita su un alfabeto Σ'.

Algoritmo di Berry e Sethi.

1. *Data e Regex, se questa non è lineare la si linearizza;*
2. *si costruisce la quadrupla $(Ini, Fin, Dig, Null)$;*
3. *a partire da e' ottenuta dalla linearizzazione, si costruisce l'automa locale A;*
4. *si rimuove la ridenominazione dalle etichette degli archi, ottenendo un'NFA ricordante e;*
5. *si procede con la subset construction.*

Esempio.
Sia data l'espressione regolare $e = ((ab + a)^* ba^*)$, si costruisca il DFA che riconosca tale Regex.

1. Poiché ovvio che e non è lineare, si procede alla sua linearizzazione. Da ciò segue $e' = (a_1 b_1 + a_2)^* b_2 a_3^*$.
2. Si procede con la costruzione della quadrupla, da cui
   - $Ini = \{a_1, a_2, b_2\}$;
   - $Fin = \{b_2, a_3\}$;
   - $Dig = \{a_1b_1, b_1a_2, a_2a_1, b_1a_1, a_2a_2, b_1b_2, a_2b_2, b_2a_3, a_3a_3\}$;
   - $Null = \varnothing$.
3. Si procede con la costruzione dell'automa, da cui si ha quanto segue.

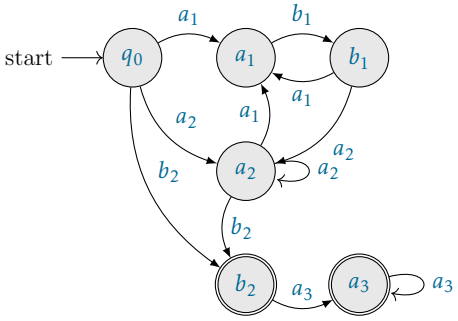
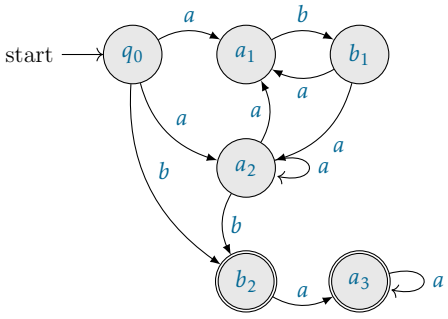


Figura 3: Automa riconoscente l'espressione $e = ((ab + a)^* ba^*)$, prima della della decodifica.

4. Si decodifica l'automa ottenuto, nel caso in esame quello di Figura (2).



5. Si conclude con la subset construction.

– 6.8 – Costruzione di Regex dato un DFA.

Si è finora detto che $\forall e \mid e \in REG, \exists A \mid L(A) \in REC$. Si descriverà ora il processo inverso ossia: dato un DFA A da questi è possibile costruire una Regex equivalente.

Teorema di Kleene.
Se $L = L(A)$ per un DFA A, allora $\exists e \mid L = L(e)$.
Più in generale si può dimostrare che

$$REC = REG$$

Osservazione.
Se il DFA A ha stati finali $\{f_1, f_2, \ldots, f_n\}$, si possono considerare n automi tali che

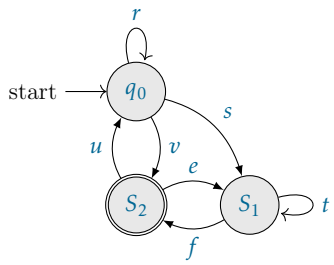
$$\forall i \in \{1, 2, \ldots, n\} \quad f_i \in F_{A_i}$$

inoltre

$$L = \bigcup_{i=1}^{n} L(A_i)$$

Si considererà dapprima un esempio, successivamente si darà una descrizione algoritmica della procedura da utilizzare.

Esempio.
Si A il seguente DFA, si trovi l'espressione regolare equivalente.



Se si elimina lo stato S_1, aggiornando opportunamente le transizioni tra q_0 e S_2 segue l'automa di Figura (4).

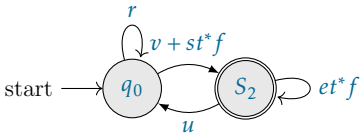
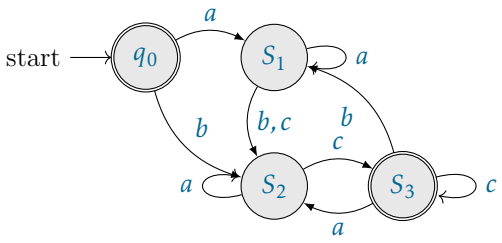


Figura 4: DFA minimale ottenuto dall'eliminazione di S_1.

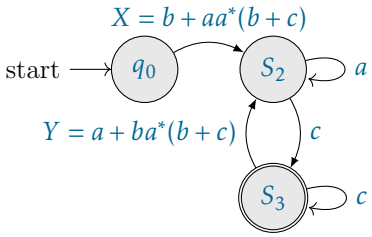
Algoritmo di Kleene.
Dando ora una descrizione algoritmica della procedura applicata nell'esempio di cui sopra:

- *Se l'automa ha più stati finali, si creano* A_i *copie,* $i \in \{1, 2, \dots, n\}$*, ciascun con un solo stato finale.*
- *Per ciascun* A_i*:*
  - *si eliminano, uno ad uno, gli stati intermedi;*
  - *si procede finché si ottiene un automa con un solo stato iniziale e un solo stato accettante;*
  - *si determina* e_i*;*
- $e = e_1 + e_2 + \cdots + e_n$

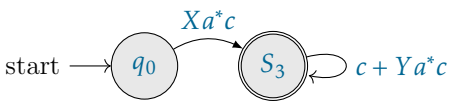
Esercizio.
Sia A il seguente DFA, si determini la Regex corrispondente.



Si parta col considerare la copia in cui l'unico stato accettante sia q_0, poiché non esistono percorsi che da q_0 riportino allo stesso, segue $e_1 = \varepsilon$. Si consideri ora la copia in cui S_3 sia l'unico stato accettante: si procede eliminando prima S_1 da cui si ottiene



da cui eliminando S_2 segue



Quindi $e_2 = Xa^*c(c + Ya^*c)$, da cui, poiché $e_1 = \varepsilon$, segue $e = e_2$.

– 7 – Proprietà dei linguaggi regolari.

Siano A, B due DFA, allora $A = B \iff L(A) = L(B)$.

– 7.1 – Minimizzazione di DFA.

Conseguenza importante di quanto detto sopra risulta essere la possibilità di minimizzare un DFA: cioè la possibilità di trovare un DFA equivalente che abbia però un numero inferiore di stati.
Si può dimostrare che per ogni DFA esiste un unico DFA minimale, cosa non vera per gli NFA.

Dato un DFA A a n stati, esistono vari algoritmi per la minimizzazione ta i quali:

- algoritmo di Moore: $\mathcal{O}\left(n^2\right)$;
- algoritmo di Hopcroft: $\mathcal{O}(n \log n)$;
- algoritmo di Brzozowski: esponenziale.

– 7.1.1 – Relazione di indistinguibilità.

Sia A un DFA, e siano p e q suoi stati, allora

$$p \mid q \iff \overline{\delta}(p, w) \in F \wedge \overline{\delta}(q, w) \in F, \forall w \in \Sigma^*$$

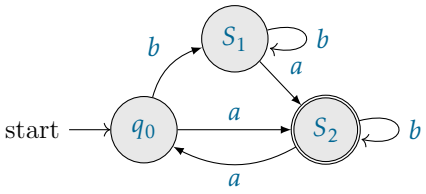
Cioè p è equivalente (*indistinguibile*) a q se, comunque presa $w \in \Sigma^*$, calcolando $\overline{\delta}$ su p e q per w, entrambe conducono a stati accettanti.

Viceversa se presi p e q stati di A, si ha

$$\overline{\delta}(p, w) \in F \wedge \overline{\delta}(q, w) \in F^C, \text{ per } w \in \Sigma^*$$

allora questi si dicono distinguibili.

Esempio.
Si consideri il seguente automa



si osserva che q_0, S_1 sono indistinguibili, mentre q_0, s_2 sono distinguibili.

– 7.1.2 – Algoritmo per il calcolo di stati equivalenti.

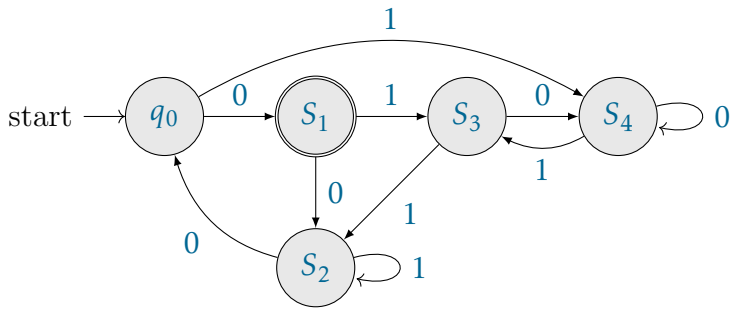
Algoritmo riempi-tabella.
Prima di procedere con la minimizzazione, è necessario identificare le coppie di stati distinguibili di un dato DFA.

Base.
Se p è accettante e q non lo è, allora $\{p,q\}$ sono distinguibili.

Induzione.
Siano p,q stati tali che dato in input α, $\delta(p,\alpha)$ e $\delta(q,\alpha)$ sono stati noti come distinguibili, allora $\{p,q\}$ sono distinguibili.

Teorema 7.1.
Se due stati non sono distinti dall'algoritmo riempi-tabella, allora sono equivalenti.

Esempio.
Sia dato il seguente automa, si stabilisca se questi è minimale oppure se minimizzabile.



Si parta con l'applicare l'Algoritmo riempi-tabella

S_3	x			
S_2	x	x		
S_1	x	x	x	
q_0	x	x	x	x
	S_4	S_3	S_2	S_1

da questi si osserva che non vi sono stati indistinguibili, da cui segue che l'automa è di per sè minimale.