**1. AUTOMI**

**Linguaggio delle soluzioni:**

Un ***cammino*** è rappresentato da qualche ***x {l,p,c,n}\****.

|  |
| --- |
| **{x {l,p,c,n}\* | iniziando in stato start e seguendo transizioni di x, terminano nello stato finale}** |
| **Nota. Il linguaggio delle soluzioni è infinito.** |

Da un ***Problema*** possiamo ricavarne un ***Linguaggio***:

* ***Problema***: trovare sequenza di mosse che permette di trasportare capra, cavolo e lupo;
* ***Linguaggio***: insieme delle stringhe sull’alfabeto {l,p,c,n}. Le stringhe corrispondono a sequenze di mosse. Tra esse cerchiamo una stringa che corrisponde ad una soluzione del problema.

|  |
| --- |
| Un ***Automa Completamente Specificato***, fornisce una procedura computazionale per decidere se una stringa è una soluzione o meno:   * Si parte dallo ***stato*** ***start***; * Si segue una ***transizione*** per ogni simbolo di input; * Se alla fine si arriva in uno ***stato obiettivo*** (accettante) accetta, altrimenti rifiuta l’input. |

**1.1 AUTOMI FINITI**

Modello semplice di calcolatore avente una quantità finita di memoria. È noto come ***macchina a stati finiti*** o ***automa finito***.

*Idea di base del funzionamento*:

* Input= stringa w su un alfabeto
* Legge i simboli di w da sinistra a destra;
* Dopo aver letto l’ultimo simbolo, l’automa indica se accetta o rifiuta la stringa input w.

**1.1.1 AUTOMA FINITO DETERMINISTICO (DFA)**

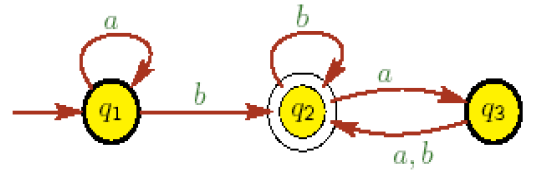
|  |
| --- |
| Sia ***A*** un ***automa finito deterministico*** (***DFA***) è 5-tupla, M = (Q, , f, q1, F), dove:  1. Q è ***insieme finito*** di stati.  2.  è ***alfabeto***, e il DFA processa stringhe su .  3. f: Q × -> Q è la ***funzione di transizione***.  4. q1 Q è lo ***stato start (o iniziale)***.  5. F (sottoinsieme di Q) è l'insieme di ***stati accettanti*** (o ***finali***). |
| *Nota. DFA è chiamato anche automa finito (FA).* |

**Funzione di transizione f: Q × -> Q** :

Definisce le regole per il cambio di stato. Se l’automa finito ha un arco da uno stato q1 a uno stato q2, etichettato col simbolo di input b, questo significa che se l’automa è nello stato q1 e legge una b, allora si muove nello stato q2.

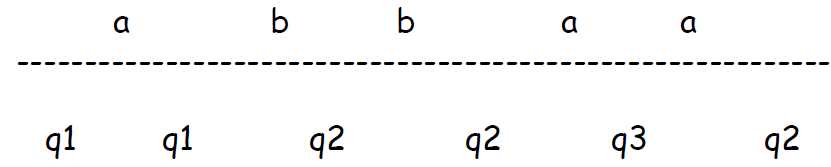
Possiamo indicare la stessa cosa con la funzione di transizione, dicendo che f(q1, b) = q2, dove q1 ϵ Q ed b ϵ Σ.

Se esiste almeno un arco uscente da q1 con label b, allora la macchina è ***deterministica***, ovvero una macchina è deterministica se il suo comportamento è specificatamente definito.

*Esempio*:

Possiamo descrivere M1 formalmente ponendo M1 = (Q, , f, q1, F), dove:

1. Q = {q1, q2, q3};
2. = {a, b}
3. f è descritto come segue:
4. q1 è lo stato start(ha freccia entrante da esterno);
5. F = {q2} (doppio cerchio)

DFA con alfabeto ={a, b}:

* Stringa arriva in input
* DFA legge 1 simbolo alla volta dal primo all’ultimo
* DFA accetta o rifiuta la stringa

*Esempi di input*: la stringa abaa è accettata se e solo se esiste una sequenza di stati q1, q1, q2, q3, q2.

**Definizione formale del funzionamento di unautoma**:

|  |
| --- |
| Sia ***M = (Q, , f, q0, F),*** consideriamo la stringa ***w = w1 w2 … wn su*** , dove ogni wi in .  Allora ***M accetta w*** se esiste una ***sequenza di stati r0, r1, . . . , rn*** in Q con tre condizioni:   1. **r0= q0** (primo stato della sequenza è quello iniziale) 2. **rn F** (ultimo stato in sequenza è uno stato accettante) 3. **f(ri, wi+1) = ri+1, per ogni i = 0,1, 2, . . . , n-1** (sequenza di stati corrisponde a transizioni valide per la stringa w).   Verificate le tre condizioni, si può affermare che M ***riconosce il linguaggio*** A, se A = {w | M accetta w}. |

**Linguaggio della Macchina*:***

|  |  |
| --- | --- |
| Se L è l'insieme di tutte le stringhe che la macchina M accetta, allora si dice che L è il ***linguaggio della macchina M*** e scriviamo ***L(M)=A***, e diciamo che ***M riconosce L***. | |
| Un linguaggio è chiamato un ***linguaggio regolare*** se un automa finito DFA lo riconosce. | |
| *Esempio1*:  L(M) è l’insieme di tutte le stringhe sull’alfabeto {a,b} della forma: {a}\*{b}{b,aa,ab}\* |  |
| *Esempio2*:  Si consideri il seguente DFA M1 con alfabeto = {0, 1}:  ***Osservazione***: L(M1) è il Linguaggio di stringhe su in cui il numero totale di 1 è dispari. |  |
| *Esempio3*:  DFA M2 con alfabeto = {0, 1}:  ***Nota***: L(M2) è Linguaggio su A che ha lunghezza 1, cioè L(M2) = {w in A\* | |w| = 1}  Si ricordi che C(L(M2)), il complemento di L(M2), è l'insieme di stringhe su A che non sono in L(M2).  *Esempio3.1:*  Si consideri il seguente DFA M3 con alfabeto A= {0, 1}  L(M3) è il Linguaggio su che ***non*** ha lunghezza 1, più di uno stato accetta, stato start anche stato accetta.  DFA accetta se e solo se lo stato start è anche lo stato accetta. |  |

**Funzione di transizione estesa*:***

|  |
| --- |
| La ***funzione di transizione f*** può essere ***estesa*** ad f\* che opera su stati e *stringhe* (invece che su stati e simboli). In generale, è possibile definirla induttivamente:  f\*(q,) = q  f\*(q, xa) = f(f\*(q, x), a)  Il linguaggio accettato da M è quindi: **L(M) = {w: f\*(q0, w) ∈ F}** |

Informalmente, la funzione di transizione estesa dà come risultato lo stato dell’automa dopo aver letto tutti i simboli di una stringa, partendo da un dato stato dell’automa.

*Esempio*:

Si consideri il DFA M1 precedentemente definito. La funzione di transizione estesa f\* è:

- f\*(q1, ε) = q1;

- f\*(q1, 001) = f(f\*(q1, 00), 1) = f(q1, 1) = q2; // si è proceduto a ritroso

- f\*(q1, 00) = f(f\*(q1, 0), 0) = f(q1, 0) = q1; // si è proceduto a ritroso

- f\*(q1, 0) = f(f\*(q1, ε), 0) = f(q1, 0) = q1.

**1.1.2 COMPLEMENTO DFA**

In generale, dato un DFA M per Linguaggio L, possiamo costruire un DFA M’ per C(L) da M:

* rendendo tutti gli stati accetta in M non-accetta in M’;
* rendendo tutti stati non-accetta in M stati accetta in M’.

|  |
| --- |
| Se si ha il linguaggio L su alfabeto Σ, ha un DFA M= (Q, Σ, f, q1, F), allora il DFA per il ***complemento*** di C(L) è: ***M’=(Q, Σ, f, q1, Q-F).*** |

|  |  |  |
| --- | --- | --- |
| *Esempio1:*  Si consideri il DFA M4 con alfabeto Σ={a, b}:    L(M4): Linguaggio di stringhe su che terminano con *bb*, sarà:  L(M4) = {w in Σ\*| w = ***s***bb per qualche stringa ***s***} | *Esempio2*:  Si consideri il DFA M5 con alfabeto ={a, b}:    L(M5) = {w|w=saa o w=sbb per una stringa s} | *Esempio3*:  Si consideri il DFA M8 con alfabeto Σ={a, b}:  Ogni a muove verso destra o sinistra.  Ogni b muove verso l’alto o il basso  DFA riconosce il Linguaggio di stringhe su Σ con numero pari di a e numero pari di b. |

**Particolari casi di automi**:

|  |  |
| --- | --- |
| Si consideri il seguente DFA M6 con alfabeto Σ ={a,b}:    Accetta tutte le possibili stringhe su Σ, cioè L(M6) = Σ\*  **In generale, ogni DFA in cui tutti gli stati sono stati accetta allora riconosce il linguaggio Σ\*.** | Si consideri il seguente DFA M6 con alfabeto Σ ={a,b}:    DFA non accetta stringhe su Σ, cioè L(M7) =  In generale, un DFA può non avere stati accetta. |

**1.2 OPERAZIONI REGOLARI (SU LINGUAGGI)**

|  |
| --- |
| Siano A e B linguaggi:   * ***Unione***: A B = {w | w A o w B} * ***Concatenazione***: AB = {vw | v A, w B} * ***Kleene*** ***star***: A\*= {w1 w2 · · · wk | k ≥ 0 e ogni wi  A} |
| **Nota**: Una collezione S di oggetti è ***chiusa*** per un’operazione f se applicando f a membri di S, f restituisce oggetto in S. |

*Esempio*:

N = {0, 1, 2, . . .} chiuso per addizione (1+2= 3∈N), non per sottrazione (1-2= -1∉N)

**TEOREMA**:

|  |
| --- |
| L’ insieme dei linguaggi regolari è ***chiuso*** per l’operazione di ***complemento***. |

Abbiamo visto che dati DFA M1 per Linguaggio L, possiamo costruire DFA M2 per Linguaggio complemento L’:

* Rendi tutti stati accetta in M1 in non-accetta in M2.
* Rendi tutti stati non-accetta in M1 in accetta in M2.

Quindi L regolare 🡪 C(L) regolare.

**TEOREMA**:

|  |
| --- |
| La classe dei ***linguaggi regolari*** è ***chiusa*** per ***l’unione***, cioè, se L1 e L2 sono linguaggi regolari, allora lo è anche L1 L2. |

**Idea**:

* L1 ha DFA M1
* L2 ha DFA M2

Una stringa *w* è in L1 ∪ L2 sse w è accettata da M1 oppure M2. Serve DFA M3 che accetta w sse w accettata da M1 o M2. Costruiamo M3 tale:

* Accetti un input esattamente quando M1 o M2 lo accetterebbero;
* Deve tener traccia di dove l’input si trovi contemporaneamente in M1 ed in M2.

**Dimostrazione**:

Supponiamo che M1 riconosca L1, dove M1= (Q1,, f1, q1, F1) ed M2 riconosca L2, dove M2= (Q2,, f2, q2, F2).

Costruiamo M3 che riconosce L1 L2, dove M3 = (Q3,, f3, q3, F3):

1. Q3= Q1×Q2= {(x, y) | x Q1, y Q2}; (*Prodotto cartesiano di Q1 e Q2*)
2. Alfabeto di M3 è , ovvero lo stesso di M1 ed M2. (*Se gli alfabeti sono diversi, era: = 1 ∪ 2*)
3. M3 ha funzione di transizione f3: Q3× -> Q3, tale che per ogni x in Q1 e y in Q2, a in : f3((x, y), a) = (f1(x, a), f2(y, a));
4. Lo stato start di M3 è q3= (q1, q2) in Q3.
5. L’ insieme di stati accetta di M3 è F3= {(x, y) in Q3| x in F1 o y in F2}

Poiché Q3= Q1×Q2, il numero di stati in M3 è |Q3| = |Q1| · |Q2|.

Quindi, |Q3| è finito poiché |Q1| e |Q2| sono finiti.

*Esempio*:

|  |  |
| --- | --- |
| Si considerino i seguenti DFA e linguaggi su = {a, b} :   * DFA M1 riconosce linguaggio A1 = L(M1) * DFA M2 riconosce linguaggio A2 = L(M2) |  |
| Il seguente è un DFA per l’unione di A1 e A2. In particolare, si procede nel seguente modo.  1. Otteniamo gli stati dal prodotto cartesiano di Q1 e Q2.  In particolare, lo stato iniziale è la coppia costituita dagli stati iniziali dei due automi;  2a. Se ci si trova nello stato (q1, r1) e si legge a, allora si nota che il primo automa resta nello stato q1, mentre il secondo automa passa allo stato r2, quindi il risultato della funzione di transizione è lo stato (q1, r2) ed il nuovo automa passa in quest’ultimo stato;  2b. se ci si trova nello stato (q1, r1) e si legge b, allora si nota che il primo automa passa allo stato q2, mentre il secondo automa passa allo stato r3, quindi il risultato della funzione di transizione è lo stato (q2, r3) ed il nuovo automa passa in quest’ultimo stato; … 3. Otteniamo gli stati finali come coppie dove è presente o lo stato finale dell’automa A o lo stato finale dell’automa B (o entrambi).  In questo esempio, gli stati finali sono: (q2, r1), (q2, r2), (q2, r3), (q1, r3). |  |
| **Nota**: Lo stato (q2, r2) non è mai raggiungibile da alcuno stato dell’automa (compreso lo stato iniziale), per cui è possibile ometterlo. | |

**TEOREMA**:

|  |
| --- |
| La classe dei linguaggi regolari è ***chiusa*** per ***l’intersezione***. Cioè, se L1 e L2 sono **linguaggi regolari**, allora lo ***è L1 L2***. |

**Idea**:

* L1 ha DFA M1
* L2 ha DFA M2

Una stringa *w* è in L1  L2 sse w è accettata sia da M1 che M2. Serve un DFA M3 che accetta w sse w è accettata da M1 e M2: deve sapere se ambedue gli automi accettano la stringa w in input, e per ottenere ciò è possibile porre ***in parallelo*** M1 e M2. Costruiamo M3 tale:

* Accetti un input esattamente quando M1 e M2 lo accetterebbero;
* Deve tener traccia di dove l’input si trovi contemporaneamente in M1 ed in M2.

**Dimostrazione:**

A tal punto, siano L1 e L2 definiti su uno stesso alfabeto Σ. Supponiamo che il DFA M1 riconosce L1, dove M1 = (Q1, Σ, f1, q1, F1), e che il DFA M2 riconosce L2, dove M2 = (Q2, Σ, f2, q2, F2). Costruiamo il DFA M3 = (Q3, Σ, f3, q3, F3) in questo modo:

- Q3 = Q1 × Q2 = {(x, y) | x in Q1, y in Q2};

- l’alfabeto di M3 è Σ;

- M3 ha f3 : Q3 × Σ → Q3 tale che per ogni x in Q1, y in Q2, a in Σ: f3((x, y), a) = (f1(x, a), f2(y, a));

- lo stato iniziale di M3 è q3 = (q1, q2) in Q3;

- l’insieme di stati accetta di M3 è F3 = {(x, y) in Q3 | x in F1 e y in F2}.

M3 è un DFA che riconosce l’intersezione. Poiché Q3 = Q1 × Q2, allora il numero di stati in M3 è |Q3| = |Q1| ∙ |Q2|. Quindi, |Q3| è finito poiché |Q1| e |Q2| sono finiti.

*Esempio*:

|  |  |
| --- | --- |
| Presi gli stessi automi, linguaggi ed alfabeto dall’ esempio mostrato in precedenza, si nota che un automa che riconosce l’**intersezione** di A1 e A2 cambia soltanto gli stati finali rispetto all’unione. Bisogna, cioè, ottenere un automa che accetti stringhe che verrebbero accettate sia da M1 sia da M2: quindi, gli stati finali sono le coppie dov’è presente sia lo stato finale di A1 sia lo stato finale di A2; lo stato finale è, in questo caso, (q2, r3). |  |

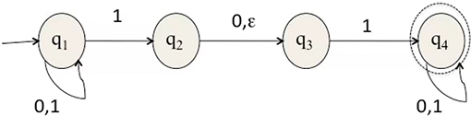
***TEOREMA***:

|  |
| --- |
| La classe dei linguaggi regolari è ***chiusa*** per la ***concatenazione***. Cioè, se A1 e A2 sono linguaggi regolari, allora lo è anche A1A2. |

Per costruire un DFA che accetti la concatenazione di stringhe appartenenti ad automi diversi, lo si fa con un nuovo tipo di macchina, ovvero:

**1.3 AUTOMI FINITI NON DETERMINISTICI**

Il ***non determinismo è una generalizzazione del determinismo***, quindi ogni automa finito deterministico è automaticamente un automa finito non deterministico.

In un DFA, lo stato successivo occupato in corrispondenza di un dato input è unicamente determinato: quindi, le macchine sono deterministiche. La funzione di transizione in un DFA è f: Q × Σ → Q, che restituisce sempre un singolo stato.  
Gli automi finiti non deterministici (NFA) permettono più scelte per il prossimo stato per un dato input. Per uno stato q, l’NFA può:  
- avere più archi uscenti da q labellati con lo stesso simbolo a, per i vari a ∈ Σ;  
- prendere ε-edge senza leggere simboli in input.

La ***differenza*** tra un automa deterministico e non è che:

* Lo stato q1 ha un arco uscente per 0, ma ne ha 2 per 1;
* Lo stato q2 ha un arco per 0 ma non è ha alcuno per 1;

Gli Automi finiti non deterministici hanno uno stato che può avere 0 o più archi uscenti per ogni simbolo dell’alfabeto.

Questo NFA ha un arco con etichetta . In generale, un NFA può avere archi etichettati con elementi dell'alfabeto o . Zero, uno, o più archi possono uscire da ciascuno stato con l'etichetta .

**Osservazione**:

Se NFA è in stato con più scelte, allora la macchina si divide in più copie di sé stessa, ognuna di essa continua una computazione in maniera indipendente dalle altre.

NFA può essere un insieme di stati, invece di un singolo stato. NFA segue ogni possibile computazione in parallelo e al termine dell’input:

* se una copia giunge in stato accetta, NFA accetta la stringa;
* se nessun cammino giunge in stato accetta, allora NFA non accetta la stringa input.

Se in stato con -transition, senza leggere input,

* NFA si divide in più copie, ognuna segue una possibile transizione,
* ogni copia continua indipendentemente da altre copie,
* NFA segue ogni possibile cammino in parallelo.
* NFA continua non deterministicamente come prima

|  |  |
| --- | --- |
| *Esempio1*:  Sia definito il seguente NFA N1 con alfabeto A = {0, 1}.  Si nota subito che lo stato iniziale q1, quando riceve in input il simbolo 1, può sia restare nello stato q1 che passare agli stati q2 e q3: questo perché quando l’automa è nello stato q2 è possibile che venga ricevuta in input una ε vuota (una tale transizione prende il nome di epsilon-transition), di conseguenza si passa direttamente anche allo stato q3. Inoltre, può capitare che in uno stato l’automa non possa accettare un simbolo dell’alfabeto (in questo caso, nello stato q3 non è possibile ricevere una label 0).  Se si riceve in input 10110, questa è la computazione che l’automa effettua 🡪 |  |
| *Esempio2*:  Dato l’NFA posto a lato, è possibile notare che esso accetta stringhe ε, a, aa, baa, baba, …;  non accetta, invece, stringhe b, ba, bb, … |  |

|  |
| --- |
| Per alfabeto , sia **Σε** ottenuto da ***aggiungendo*** . Cioè, **Σε = Σ ∪ {ε}**. |
| Un automa finito non-deterministico NFA A è una 5-tupla M= (Q,, f, q0, F), con  1. Q è l’insieme finiti di stati  2. ∑ è un alfabeto, e il DFA processa stringhe su Σ;  3. f: Q × ∑ε -> P(Q) funzione di transizione, dove P(Q) è l’insieme di tutti i sottoinsiemi possibili di Q;  4. q0∈Q è stato start  5. F⊆Q è insieme di stati accettanti. |
| **Nota**. La differenza tra DFA e NFA è nella funzione di transizione f, la quale:   * ammette mosse tipo * restituisce ***insieme*** di stati invece di un solo stato. |

|  |  |
| --- | --- |
| *Esempio1*:  Sia definito l’NFA N = ({q0, q1, q2}, {0, 1}, δ, q0, {q2}), posto di seguito, dove δ è la funzione di transizione descritta nella tabella a lato. |  |
| *Esempio2*:  Sia definito l’NFA N2 = ({q1, q2, q3, q4}, {0, 1}, δ, q1, {q4}), posto di seguito, dove δ è la funzione di transizione descritta nella tabella a lato. |  |
| **Nota**: Si ricordi che, una volta che l’automa è nello stato q2, esso può passare allo stato q3 in quanto è presente una ε-transition in tale stato. | |

**Computazione di NFA**:

Sia N = (Q, , f, q0, F) un NFA e sia *w* una stringa sull’alfabeto , allora N ***accetta*** *w* se possiamo scrivere *w* come *w* = y1, y2, …, ym, dove ogni yi è un elemento di ed esiste una sequenza di stati r0, r1, ..., rm in Q con tre condizioni:

1. r0 = q0 (La macchina inizia nello stato iniziale)
2. ri+1 in f(ri, yi+1) per ogni i=0, 1, 2, …, m-1 (Lo stato ri+1 è uno dei possibili stati successivi quando N è nello stato ri e sta leggendo yi+1)
3. rm in F (La macchina accetta il suo input se l’ultimo stato è uno stato accettante)

Informalmente, vuol dire che N accetta w se esiste una possibile computazione che porta allo stato di accettazione. Si noti che la stringa generica yi non appartiene a Σ, bensì appartiene a Σε in quanto è possibile che essa sia un ε per effettuare una epsilon-transition; è questa l’unica differenza tra la computazione di un DFA e la computazione di un NFA.

|  |
| --- |
| L’insieme di stringhe accettate da NFA N è il linguaggio ***riconosciuto*** da N ed è denotato con L(N). |
| **Nota**: Ogni DFA è anche un NFA. Infatti, vedremo che è possibile esprimere un NFA attraverso un corrispondente DFA, e che i primi sono semplicemente un modo più compatto per descrivere i secondi. |

|  |  |
| --- | --- |
| Sia definito l’NFA N posto a lato. Allora esso accetta il linguaggio {x01 | x ∈ Σ\*}.  Ad esempio, se l’input è 01001 abbiamo il seguente albero di computazione:  - 0, da q0 si passa sia a q0 che a q1;  - 1, da q0 si passa nuovamente a q0, e da q1 si passa a q2 (in quest’ultimo stato, la computazione si arresta);  - 0, da q0 si passa sia a q0 che a q1;  - 0, da q0 si passa sia a q0 che a q1, e da q1 con 0 si arresta la computazione;  - 1, da q0 si passa a q0 e da q1 si passa a q2, e in quest’ultimo stato la stringa viene accettata. |  |

**1.3.1 EQUIVALENZE DI DFA E NFA**

|  |
| --- |
| Due macchine sono ***equivalenti*** se ***riconoscono lo stesso linguaggio***. |
| **TEOREMA**: Per ogni automa finito non deterministico NFA esiste un automa finito deterministico DFA ***equivalente***.  Ogni NFA N ha un equivalente DFA M, cioè se N è un NFA, allora esiste DFA M t.c. ***L(M) = L(N).*** |

**Idea**:

Trasformare un NFA in un DFA equivalente che simula il NFA. Si dovrà capire quali saranno lo stato iniziale e quelli accettanti del DFA e la sua funzione di transizione.

**Nota***: Se k è il numero degli stati dell’NFA, esso avrà 2k sottoinsiemi di stati. Ogni sottoinsieme corrisponde ad una delle possibilità che il DFA deve ricordare, quindi il DFA che simula l’NFA avrà 2k stati.*

**Dimostrazione*:***

|  |  |  |
| --- | --- | --- |
| Costruiamo, a partire da N, il DFA M equivalente. L’idea è simile a quella usata per costruire l’automa che riconosce l’unione di due linguaggi: simuliamo tutte le computazioni fatte dal NFA in parallelo. Consideriamo il seguente automa. |  | |
| L’automa deterministico dovrà, partendo dallo stato iniziale in presenza di input 1, ricordare all’interno dello stato che andrà ad occupare che l’automa non deterministico può essere in ognuno degli stati q1, q2 e q3: è possibile ottenere ciò andando a considerare uno stato che è dato dall’insieme degli stati {q1, q2, q3}. Allo stesso modo, in presenza di input 0, seguendo la prima computazione allora si passa da q1 a q1, mentre per la seconda si passa da q2 a q3, e la terza computazione si arresta: questo si può considerare nell’automa deterministico come uno stato dato dall’insieme degli stati {q1, q3}. Si procede in questo modo per ogni possibile computazione. Otteniamo, quindi, il seguente DFA. Si osservi che dallo stato {q1, q2, q3}, in presenza di input 0, si passa allo stato {q1, q3}. Gli stati finali del DFA saranno tutti quelli in cui compare *almeno uno* tra gli stati finali del NFA: in questo caso, solo gli stati in cui compare q4. |  | |
| Formalizzando, sia L = L(N) per un NFA N = (QN, Σ, fN, qN, FN); allora, costruiamo un DFA D = (QD, Σ, fD, qD, FD).  Partiamo da un automa D’ che non considera le ε-transition, dove, per ogni r ∈ Q e a ∈ Σ: - Q = P(QN), ossia ogni stato in D’ sarà un sottoinsieme dell’insieme potenza degli stati del NFA N; - f(r, a) = , ossia la funzione di transizione darà come risultato l’unione dei risultati (per ogni elemento r dell’insieme R), della funzione di transizione fN(r, a); - q = {qN}, ossia lo stato iniziale sarà l’insieme in cui compare lo stato iniziale del NFA N; - f = {r ∈ Q | r ∩ FN ≠ ∅}, ossia gli stati finali saranno tutti quegli stati che contengono al loro interno uno stato finale del NFA N.  Per ogni stato in fN, se vi è una ε-transition allora dobbiamo considerarla. A tale scopo, definiamo l’insieme **E(r)** = r ∪ {q | q è raggiungibile da uno stato in r con 1 o più archi labellati ε}, cioè l’insieme che considera l’unione tra uno stato di D’ (in quanto r ∈ Q) e l’insieme degli stati raggiungibili da uno stato in r che ammettono almeno una ε-transition.  Per la funzione di transizione estesa si ha che f\*N(q1, 10110) = {q1, q2, q4}.  Formalmente, si ha che: - , quindi non ci sono lettere lette ed applichiamo la sola epsilon-transition; - , quindi si prendono tutti gli stati in cui potrebbe trovarsi l’automa dopo aver letto la stringa x ∈ Σ\*, e per ognuno di questi stati applichiamo la funzione di transizione del NFA, applichiamo poi la funzione E, facciamo l’unione di ciò che abbiamo ottenuto ed otteniamo l’insieme di stati raggiungibili quando l’input è la stringa xa.  Risulta, per ogni x ∈ Σ\*, , cioè che le funzioni di transizione estese del NFA e del DFA corrispondente coincidono (ciò significa che i due automi accettano le stesse stringhe); questo risultato si può dimostrare induttivamente. Quindi, sappiamo che D simula N su ogni input x. Inoltre, D accetta x se e solo se N accetta x. Infine, il linguaggio L = L(N) = L(D). Ciò significa che D e N sono equivalenti.  **Dimostrazione**:  Mostriamo per induzione su |w|, ponendo q0 = qN, che . *Base:* w = ε. L’enunciato segue dalla definizione. *Passo*: Supponiamo che l’uguaglianza è vera per una certa stringa x, e consideriamo una stringa xa:  (per definizione)  (per ipotesi induttiva)  (per definizione)  . | | |
| *Esempio1*:  Considerando l’esempio precedente, si ha ad esempio che E({q1}) = {q1}, e E({q2}) = {q2, q3}: questo perché q1 non accetta epsilon-transition, a differenza di q2. Quindi, avremo il DFA D: - QD = P(QN), che coincide con quanto definito per D’; - fD(r, a) = , ossia la funzione di transizione darà come risultato l’unione dei risultati (per ogni elemento r dell’insieme R), della E della funzione di transizione fN(r, a); - qD = E({qN}), ossia lo stato iniziale sarà la E dell’insieme in cui compare lo stato iniziale del NFA N; - FD = {r ∈ QD | r ∩ FN ≠ ∅}, che coincide con quanto definito per D’.  Considerando l’esempio precedente, si ha ad esempio che fN(q1, 1) = {q1, q2} se non consideriamo le ε-transition.  Se consideriamo le ε-transition, invece, si ha che E(fN(q1, 1)) = {q1, q2, q3}. | | |
| *Esempio2*:  Considerando l’automa posto a lato, otteniamo che il suo stato iniziale sarà E({r1}) = {r1, r3}.  Abbiamo, inoltre, il suo DFA equivalente:  - {r1, r3}, con input ‘a’ si ha che da r1 non si passa ad alcuno stato, mentre da r3 si passa allo stato r1. A questo punto, si applica nuovamente la funzione E (in quanto r1 accetta una ε-transition) e si passa ancora dallo stato r1 allo stato r3. Il prossimo stato sarà, quindi, ancora {r1, r3};  - {r1, r3}, con input ‘b’ si ha che da r1 si passa a r2, mentre da r3 non si passa ad alcuno stato. Lo stato r2 non ammette ε-transition, quindi il nuovo stato sarà {r2};  - {r2}, con input ‘a’ si ha che da r2 si passa ad r2 e da r2 si passa ad r3, quindi il nuovo stato sarà {r2, r3};  - {r2}, con input ‘b’ si ha che da r2 si passa ad r3, quindi il nuovo stato sarà {r3};  - {r2, r3}, con input ‘a’ da r2 vale il punto precedente, mentre da r3 si passa ad r1, ma in quest’ultimo caso applicando la E otteniamo che si passa anche da r1 a r3.  Di conseguenza, il nuovo stato sarà {r1, r2, r3};  - {r2, r3}, con input ‘b’ da r2 vale il punto precedente, mentre da r3 non c’è nessuna transizione. Di conseguenza, il nuovo stato sarà {r3};  - {r3}, con input ‘a’ da r3 si passa a r1, ma applicando la E otteniamo che si passa anche da r1 a r3. Il nuovo stato sarà, quindi, lo stato iniziale {r1, r3};  - {r3}, con input ‘b’ non si passa ad alcuno stato. Tuttavia, l’automa è deterministico per cui non possiamo ignorare un eventuale input, per cui si passa allo stato vuoto;  - considerando lo stato vuoto, per ogni input si passa nuovamente allo stato vuoto;  - {r1, r2, r3}, con input ‘a’ ogni stato passa a quello successivo, per cui il nuovo stato sarà ancora {r1, r2, r3};  - {r1, r2, r3}, con input ‘b’ si ha che da r1 si passa ad r2, da r2 si passa ad r3, mentre da r3 non si passa ad alcuno stato. Quindi, il nuovo stato sarà {r1, r3}.  Ora, bisogna identificare gli stati finali: sono tutti quegli stati al cui interno è presente lo stato r1. L’automa deterministico corrispondente avrà, quindi, come stati finali {r1, r3} e {r1, r2, r3}. | |  |
| **Nota:** f\*N(qN, x) = insieme di stati raggiungibili da qN su input x. Ciò vuol dire che la funzione estesa di un automa non deterministico può dare come risultato più stati, in quanto è possibile avere diverse copie di computazione. | | |

**Corollario**:

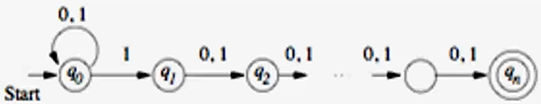
|  |
| --- |
| Un ***linguaggio è regolare*** se e solo se qualche ***automa finito non deterministico lo riconosce***. |

**Dimostrazione*:***

Se L è regolare, allora esiste un DFA che lo riconosce; ma ogni DFA è anche un NFA, quindi esiste un NFA per L.

Dal teorema precedente, ogni NFA ha un equivalente DFA. Quindi, se esiste un NFA che riconosce L, allora esiste un DFA che riconosce L.  
Formalmente: L = L(N) = L(D) ⇒ ∃ D: L(D) = L(N) = L; quindi, L è un linguaggio regolare.

Si osservi che, sebbene sia possibile ottenere un DFA equivalente da un determinato NFA, è bene (spesso) utilizzare questi ultimi perché sono più facili da rappresentare.

**Nota.** Esiste un NFA N con n+1 stati che non ha nessun DFA equivalente con meno di 2n stati.

Il linguaggio dell’automa a dx è L(N) = {x1c2c3 ∙∙∙ | x ∈ {0, 1}\*, ci ∈ {0, 1}}.  
Questo automa è simile all’automa che riconosce tutte le stringhe che terminano con 01, ma è costruito in modo tale che lo stato iniziale cicli sia con 0 che con 1, dopodiché ha una transizione con 1 da q0 a q1 (quindi si ha uno sdoppiamento della computazione); inoltre, dallo stato q1 allo stato qn si avanza con qualsiasi simbolo che riceve in input. Quindi, una stringa per essere accettata deve avere qualunque simbolo all’inizio, un 1 e poi n-1 simboli qualsiasi.

In sintesi, mentre un NFA può sdoppiarsi ad ogni stato ed avere più computazioni, un DFA deve necessariamente rappresentare gli stati in modo tale che possa ricordare gli ultimi n simboli che ha letto. Questo non è semplice da realizzare, in quanto ci sono 2n sequenze di bit (da rappresentare con opportuni stati) a1a2 ∙∙∙ an da ricordare.

**1.4 OPERAZIONI REGOLARI CON GLI AUTOMI**

Lo scopo delle operazioni regolari è quello di provare che ***unione***, ***concatenazione*** e ***kleene star*** di linguaggi regolari sono ancora regolari.

L’uso del non determinismo rende le prove molto più semplici.

**1.4.1 OPERAZIONE DI UNIONE PER GLI AUTOMI**

**TEOREMA**:

|  |  |
| --- | --- |
| La classe dei ***linguaggi regolari*** è chiusa rispetto all’operazione di ***unione***. | |
| ***Idea***:  Dati due linguaggi regolari A1 e A2 riconosciuti da N1 e N2, vogliamo provare che ***A1 ∪ A2*** ***è regolare*** componendo un nuovo NFA N.  La macchina N deve accettare il suo input se N1 o N2 accetta questo input. La nuova macchina ha un nuovo stato iniziale che si dirama negli stati iniziali delle vecchie macchine con ε-archi. In questo modo, la nuova macchina ipotizza non deterministicamente quale delle due macchine accetta l’input. Se una di essi accetta, allora anche N lo accetterà. | ***Dimostrazione***:  Sia N1 = (Q1, , f1, q1, F1) che riconosce A1 ed N2 = (Q2, , f2, q2, F2) che riconosce A2.  Costruiamo N = (Q, , f, q, F) per A1 ∪ A2:   1. Q = {q0} ∪ Q1 ∪ Q2   Gli stati di N sono tutti gli stati di N1 ed N2, con l’aggiunta di un nuovo stato iniziale q0.   1. Lo stato q0 è lo stato iniziale di N. 2. L’insieme degli stati accettanti F = F1 ∪ F2   Gli stati accettanti di N sono tutti gli stati accettanti di N1 ed N2. In questo modo, N accetta se N1 accetta o N2 accetta.   1. Definiamo f in modo che per ogni *q* in *Q* e ogni *a* in ∑ε: |

**1.4.2 OPERAZIONE DI CONCATENAZIONE PER GLI AUTOMI**

Si ricordi che, dati due linguaggi A, B regolari, la concatenazione è l’insieme AB = {vw | v ∈ A, w ∈ B}.

**TEOREMA:**

|  |  |
| --- | --- |
| La classe dei ***linguaggi regolari*** è chiusa rispetto all’operazione di ***concatenazione***. | |
| **Idea**:  Dati due linguaggi regolari A1 e A2 riconosciuti da N1 e N2, vogliamo provare che ***A1 o A2*** ***è regolare*** componendo un nuovo NFA N.  Poniamo come stato iniziale di N, lo stato iniziale di N1.  Gli stati accettanti di N1 hanno degli ulteriori ε-archi che non deterministicamente permettono di diramarsi in N2 ogni volta che N1 è in uno stato accettante, indicando che ha trovato un pezzo iniziale dell’input che costituisce una stringa in A1.  Gli stati accettanti di N sono solo gli stati accettanti di N2. Quindi, esso accetta quando l’input può essere diviso in due parti, la prima accettata da N1 e la seconda da N2. Possiamo pensare che N non deterministicamente ipotizza dove effettuare la divisione. | ***Dimostrazione***:  Sia N1 = (Q1, , f1, q1, F1) che riconosce A1 ed N2 = (Q2, , f2, q2, F2) che riconosce A2.  Costruiamo N = (Q, , f, q, F) per A1 o A2:   1. Q = Q1 ∪ Q2   Gli stati di N sono tutti gli stati di N1 ed N2.   1. L’alfabeto è lo stesso dei due linguaggi L1 ed L2. 2. Lo stato q0 è uguale allo stato iniziale di N1. 3. Gli stati accettanti F2 sono uguali agli stati accettanti di N2. 4. Definiamo f in modo che per ogni *q* in *Q* e ogni *a* in ∑ε:     La funzione di transizione *δ(q, a)*, dato un generico stato *q* ed una stringa *a*: - se *q* è uno stato non finale del primo automa, allora la transizione è quella specificata dal primo automa; - se *q* è uno stato finale del primo automa e *a* è diversa da epsilon, allora la transizione è quella specificata dal primo automa; - se *q* è uno stato finale del primo automa e *a* è ε, allora alle transizioni che già si avevano per il primo automa vanno aggiunte anche le ε-transition verso lo stato iniziale del secondo automa; - se *q* è uno stato del secondo automa, allora le transizioni sono quelle specificate dal secondo automa. |
| Analogamente, supponiamo di avere una stringa w = uv, non appartenente a L1L2: questo significa che u ∉ L1 OR v ∉ L2. Accadrà, quindi, che la stringa non verrà accettata da (almeno) uno dei sottoautomi, di conseguenza w non verrà accettata dall’automa N. | |
| La concatenazione è stata definita per due linguaggi, ma in realtà è possibile iterare ed andare a concatenare più di due linguaggi: per concatenare L1, L2, L3, ad esempio, si può prima concatenare L1 ed L2 per poi procedere con la concatenazione di L3. In generale, questo risultato si può estendere alla **Kleene-star**: **L\* = {x1x2 ∙∙∙ xk | k ≥ 0, xi ∈ L, i = 1, 2, …, k}**. | |
| *Esempio1*:  Siano L1 = {w | |w| è pari} e L2 = {w | w contiene almeno due 0}. Vogliamo costruire gli automi per questi due linguaggi, e a partire da questi ultimi un automa per la loro concatenazione L1L2. A tal scopo, sia N1 l’automa per L1, e sia N2 l’automa per L2. Otteniamo il seguente risultato.    Quindi, sia L(N) = L1L2. Dall’idea dimostrativa del teorema precedente, otteniamo il seguente automa N.  Notiamo che, ad esempio, la stringa 10100 è accettata da N, dato che è gestibile come stringa costituita dalle sottostringhe 10 ∈ L1 e 100 ∈ L2. Questa stringa è gestita dall’automa come descritto dall’albero a destra.    *Esempio2*:  Dati i linguaggi L1 ed L2 precedenti, vogliamo costruire un automa per la loro concatenazione L2L1. A tal scopo, sia N1 l’automa per L1, e sia N2 l’automa per L2. Quindi, sia L(N’) = L2L1. Le stringhe accettate devono, quindi, essere costituite da una sottostringa contenente due o più 0, e da una sottostringa ad essa concatenata di lunghezza pari. Dall’idea dimostrativa del teorema precedente, otteniamo il seguente automa N’.    *Esempio3*:  Siano L1 = {w | |w| è dispari} e L2 = {w | w contiene almeno due 0}. Vogliamo costruire gli automi per questi due linguaggi, e a partire da questi ultimi un automa per la loro concatenazione L1L2. A tal scopo, sia N1 l’automa per L1, e sia N2 l’automa per L2. Otteniamo il risultato sottostante. Ad esempio, la stringa 1010111 (gestibile come 1010 ∈ L2 e 111 ∈ L1) viene accettata. La 0101011 (gestibile come 0101 ∈ L2 e 011 ∈ L1) segue l’albero computazionale di cui a destra. | |

**1.4.3 OPERAZIONE STAR PER GLI AUTOMI**

**TEOREMA:**

|  |  |
| --- | --- |
| La classe dei ***linguaggi regolari*** è chiusa rispetto all’operazione ***star***. | |
| **Idea**:  Dato un linguaggio regolare A1 e vogliamo provare che anche A1\* è regolare.  Prendiamo un NFA N1 per A1 e lo modifichiamo per riconoscere A1\*. L’NFA N risultante accetterà il suo input quando esso può essere diviso in varie parti ed N1 accetta ogni parte. Possiamo costruire N come N1 con ε-archi supplementari che dagli stati accettanti ritornano allo stato iniziale. In questo modo, quando l’elaborazione giunge alla fine di una parte che N1 accetta, la macchina N ha la scelta di tornare indietro allo stato iniziale per provare a leggere un'altra parte che N1 accetta. Inoltre, si aggiunge un nuovo stato iniziale, che è anche uno stato accettante, e che ha un ε-arco entrante nel vecchio stato iniziale. | **Dimostrazione**:  Sia N1 = (Q1, , f1, q1, F1) che riconosce A1.  Costruiamo N = (Q, , f, q0, F) per A1\*:   1. Q = {q0} ∪ Q1   Gli stati di N sono tutti gli stati di N1 più un nuovo stato iniziale.   1. Lo stato q0 è il nuovo stato iniziale. 2. F = {q0} ∪ F1   Gli stati accettanti sono i vecchi stati accettanti più il nuovo stato iniziale.   1. Definiamo f in modo che per ogni *q* in *Q* e ogni *a* in ∑ε: |
| *Esempio1*:  Sia L = {w | |w| è pari OR w = 00x}. Un automa non deterministico che riconosca L è il seguente (N1, a sinistra), dove in rosso è posto il sottoautoma che accetta le stringhe pari, e in blu è posto l’automa che riconosce le stringhe che iniziano con 00. Notiamo che l’OR, in un automa non deterministico, è semplicemente rappresentabile aggiungendo uno stato iniziale generico collegato con una ε-transition agli “ex stati iniziali” dei sottoautomi. Questo è possibile in quanto, con i NFA, la computazione si sdoppia direttamente all’inizio e prosegue nei due sottoautomi. A destra, invece, è posto l’automa N che riconosce la Kleene-star di L, cioè L\* = L(N).    Ad esempio, la stringa 0011 (considerabile come unica stringa che inizia con due 0, oppure come stringa costituita da x1 = 00 concatenata con x2 = 11) viene accettata.  Più in generale, per la costruzione dell’OR (**unione**)supponiamo di avere due linguaggi, L1 ed L2, riconosciuti rispettivamente da due automi N1 e N2. Per costruire l’automa N, che riconosce il linguaggio L1 ∪ L2: 1. si crea un nuovo stato, che sarà lo stato iniziale dell’automa N; 2. colleghiamo lo stato iniziale di N con i due stati iniziali di N1 e N2. In questo modo, abbiamo un NFA che riconosce tutte le stringhe dell’unione, in quanto (per ogni stringa) la computazione si sdoppia ed avremo due rami computazionali: uno eseguito da N1 e uno eseguito da N2. Ora: - se la stringa appartiene all’unione, allora essa o arriva in uno stato finale di N1 o arriva in uno stato finale di N2 (o in entrambi), e viene accettata dall’automa N; - se la stringa non appartiene all’unione, allora essa non arriva in alcuno stato finale di N1 o N2, e non viene accettata dall’automa N. | |