**1 - Automi**

|  |
| --- |
| Sia ***A*** un ***automa finito deterministico*** (***DFA***) è 5-tupla, M = (Q, , f, q0, 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*** (definisce le regole per il cambio di stato). 4. q0 Q è lo ***stato start (o iniziale)***. 5. F⊆Q è l'insieme di ***stati accettanti*** (o ***finali***). |
| 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***. |
| Sia ***M = (Q, , f, q0, F),*** consideriamo la stringa ***w = w1 w2 … wn su*** , dove ogni wi in , (o in se è un NFA):  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}. |

**Complemento**:

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)***.

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

Dim:

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.

**Intersezione**:

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

Dim:

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|.

**Unione**:

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

Dim:

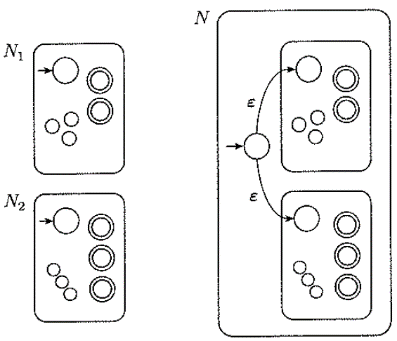
Supponiamo che M1 riconosca L1, dove M1= (Q1,, f1, q1, F1) ed M2 riconosca L2, dove M2= (Q2,, f2, q2, F2).

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, y in Q2 ed 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|.

*Oppure tramite NFA*:

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. Se una di essi accetta, allora anche N lo accetterà.

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:

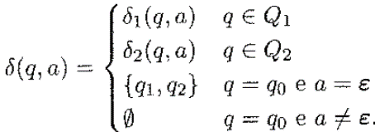
1. Q = {q0} ∪ Q1 ∪ Q2

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:



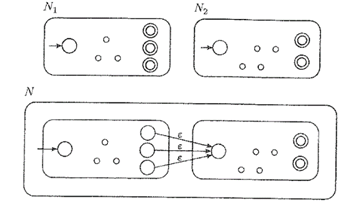
**Concatenazione**:

|  |
| --- |
| La classe dei ***linguaggi regolari*** è ***chiusa*** per la ***concatenazione***. Cioè, se L1 e L2 sono linguaggi regolari, allora lo è anche L1L2. |

Dim:

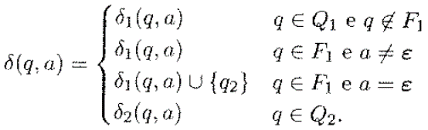
Poniamo come stato iniziale di N, lo stato iniziale di N1. Gli stati accettanti di N1 hanno degli ulteriori ε-archi che permettono di diramarsi in N2 ogni volta che N1 è in uno stato accettante. Gli stati accettanti di N sono solo gli stati accettanti di N2, esso accetta quando l’input può essere diviso in due parti, la prima accettata da N1 e la seconda da N2.

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:



**Kleene Star**:

|  |
| --- |
| La classe dei ***linguaggi regolari*** è chiusa rispetto all’operazione ***star***. |

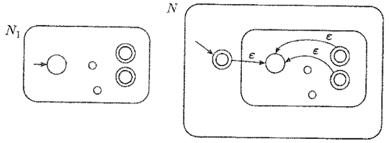
Dim:

Possiamo costruire N come N1 con ε-archi supplementari che dagli stati accettanti ritornano allo stato iniziale. 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.

Sia N1 = (Q1, , f1, q1, F1) che riconosce A1. Costruiamo N = (Q, , f, q0, F) per A1\*:

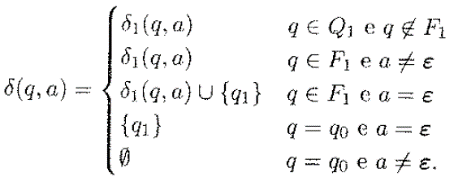
1. Q = {q0} ∪ Q1

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:



|  |
| --- |
| 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).*** |

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 dobbiamo definire 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:

- 𝑓𝑁∗(𝑞𝑁,𝜀)=𝐸({𝑞𝑁}), quindi non ci sono lettere lette ed applichiamo la sola epsilon-transition;

- 𝑓𝑁∗(𝑞𝑁,𝑥𝑎)=⋃𝐸(𝑓𝑁(𝑟,𝑎))𝑟 ∈ 𝑓𝑁∗(𝑞𝑁,𝑥), 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\*, 𝑓𝐷∗(𝑞𝐷,𝑥)=𝑓𝑁∗(𝑞𝑁,𝑥), 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.

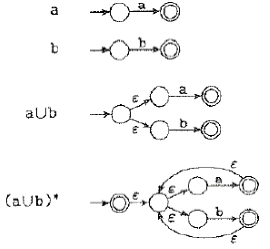
**2 – Espressione regolare**

**Definizione induttiva di espressione regolare (e.r.):**

***BASE***:  
- Se ε e ∅ sono espressioni regolari: L(ε) = {ε} e L(∅) = ∅;  
- Ed a ∈ Σ, allora a è un’espressione regolare: L(a) = {a}.  
***PASSO***: Se R1 e R2 sono espressioni regolari, allora:  
- R1 ∪ R2 è un’espressione regolare che rappresenta L(R1) ∪ L(R2);  
- R1R2 è un’espressione regolare che rappresenta L(R1)L(R2);  
- R1\* è un’espressione regolare che rappresenta (L(R1))\*.

**Teorema di Kleene**:

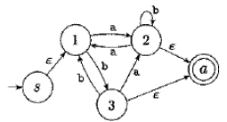
|  |
| --- |
| Un linguaggio è ***regolare*** se e solo se qualche ***espressione regolare*** lo descrive: L ↔ E.R. |
| **Lemma1**. Se un linguaggio è descritto da un’***espressione regolare***, allora esso è ***regolare***. |

Dim:

Data una espressione regolare, la si scompone in sotto-espressioni più semplici (aiutandoci con una struttura ad albero) e si costruiscono una serie di NFA che riconoscono queste sotto-espressioni semplici.

Con quest’ultimi costruiamo altri NFA, tramite operazioni regolari con gli automi, che riconoscono sotto-espressioni più complesse, fino ad arrivare a costruire un NFA che riconosca l’espressione regolare data.

|  |
| --- |
| **Lemma2**. Se un linguaggio è ***regolare***, allora è descritto da un’***espressione regolare***. |

Dim:

Dato un automa A, costruiamo un nuovo automa avente un nuovo stato iniziale senza archi entranti ed un nuovo stato finale senza archi uscenti. Si rimuove uno stato alla volta, che non siano i nuovi stati inseriti, modificando le etichette sugli archi in modo che l’automa risultante accetti le stesse stringhe, fino ad avere solo i due nuovi stati inseriti inizialmente.

**Pumping Lemma**:

|  |
| --- |
| Se ***A*** è un ***linguaggio*** ***regolare***, allora esiste un numero ***p*** (***lunghezza del pumping***) tale che ***s*** è una qualsiasi stringa in A di lunghezza almeno ***p***, allora ***s*** può essere divisa in ***tre parti***, ***s = x y z***, che soddisfa le seguenti condizioni:   1. ***Per ogni i ≥ 0, xyiz ∈ A;*** 2. ***|y| > 0;*** 3. ***|xy| ≤ p***. |

**3 - MdT**

|  |
| --- |
| Una ***macchina di Turing*** è una 7-tupla (Q, ∑, Γ, f, q0, qaccept, qreject), dove Q, ∑, Γ sono tutti insiemi finiti e:   1. ***Q*** è ***l’insieme degli stati***; 2. ***∑*** è ***l’alfabeto di input*** non contenente il ***simbolo blank*** ‘**\_’**; 3. ***Γ*** è ***l’alfabeto del nastro*** con **\_** ∈ Γ e ∑ ⊆ Γ contenente tutti i simboli che possono essere scritti all’interno di una cella di memoria (tecnicamente, l’unione tra l’alfabeto di lavoro e l’insieme dei simboli non processabili dalla macchina, come \_); 4. ***F: Q x Γ 🡪 Q x Γ x {L, R}*** è la ***funzione di transizione***, dove le lettere lette sono all’interno del nastro (in ogni istante si hanno dei simboli nelle varie celle, e la cella a cui punta la testina rappresenta lo stato q in cui si trova la macchina).; 5. ***q0*** ∈ Q è lo ***stato iniziale***; 6. ***qaccept*** ∈ Q è lo ***stato di*** ***accettazione***; |

Una **configurazione** di una Macchina di Turing è una descrizione concisa di stato e contenuto del nastro. Trattasi di una stringa C = *uqv*, dove:  
- *q* è lo *stato* occupato dalla macchina M;  
- *uv* è il *contenuto del nastro* (sinistra – destra);  
- la testina punta sul primo (cioè, più a sinistra) simbolo di *v* (su primo blank \_ se v = ε);  
- dopo *v* sono presenti solo simboli blank \_.

|  |
| --- |
| Una ***MdT M*** accetta una ***stringa w*** se esiste una ***computazione*** (***sequenza di configurazione***) di M: C1, … Ck  tale che   1. C1 = *q0w* è la configurazione iniziale di M con input w; 2. Ogni Ci produce Ci+1 per ogni i=1, …, k-1; 3. Ck è una configurazione di accept. |
| Un linguaggio si dice ***Turing-riconoscibile*** se esiste una macchina di Turing che lo riconosce. |

Formalmente, un linguaggio L si dice Turing riconoscibile se esiste una Macchina di Turing M tale che L(M) = L. Di conseguenza, la macchina accetta tutte le stringhe del linguaggio. Se, invece, una stringa w∉L, allora la stringa può essere o *rifiutata* o mandare la macchina *in loop*.

Possiamo evitare il loop costruendo le macchine che si fermano (accettando o rifiutando) *su ogni input*: trattasi dei ***deciders***.  
Si dice che un decider *decide* il linguaggio L se esso riconosce L.

|  |
| --- |
| Un linguaggio si dice ***Turing-decidibile*** o semplicemente ***decidibile*** se esiste una macchina di Turing che lo decide. |

Formalmente, un linguaggio L si dice Turing decidibile se esiste una Macchina di Turing M tale che L(M) = L e M è un ***decider***.  
La ***differenza tra un linguaggio L Turing riconoscibile e Turing decidibile*** risiede nel fatto che i primi possono mandare le MdT che li riconoscono in loop, mentre i secondi possono far sì che le MdT che li decidono o accettino o rifiutino le loro stringhe *su ogni input*.

Uno **stayer** è una Macchina di Turing la cui testina può rimanere sulla stessa cella del nastro durante una transizione; formalmente, essa è la stessa settupla M = (Q, Σ, Г, δ, q0, qaccept, qreject), con la sola differenza che la funzione di transizione è δ: Q × Г → Q × Г × {*L, S, R*}, dove **S** indica che la testina resta ferma durante la transizione (S sta per “stay”).

Una MdT **multinastro** (a k nastri) è una normale Macchina di Turing avente k nastri. Inizialmente, l’input compare sul primo nastro e gli altri sono vuoti. La sua funzione di transizione è **δ: Q × Гk → Q × Гk × {*L, S, R*}k**; dunque, essa muove (in modo indipendente) le testine dei vari nastri.  
Questa variante è simile ad una MdT convenzionale: usa k nastri, con k ≥ 1. Ciascun nastro ha una propria testina, e con una mossa si specificano (oltre al nuovo stato): i k simboli letti, i k simboli da scrivere e i k movimenti delle k testine. Come configurazione iniziale, essa avrà l’input sul primo nastro ed i rimanenti nastri saranno vuoti.

|  |
| --- |
| *MdT e MdT multinastro sono modelli equivalenti.* |

Dim:

L’implicazione diretta è ovvia, in quanto una MdT è una MdT multinastro, con k = 1. Vogliamo dimostrare che per ogni MdT multinastro è possibile costruire una MdT convenzionale che riconosce lo stesso linguaggio. Effettuiamo una simulazione, utilizziamo una MdT multinastro, con k=3.  
Una soluzione è immaginare i k nastri affiancati, ognuno con indicata la posizione della testina. Dobbiamo codificare questa informazione su un solo nastro. Per fare ciò possiamo concatenare il contenuto dei k nastri, su k blocchi consecutivi separati da un carattere particolare (con #):  
- ogni blocco avrà lunghezza variabile che dipende dal contenuto del nastro corrispondente;  
- un elemento marcato (con ˙) nel blocco i-esimo indica la posizione della testina i-esima (ad esempio, se la testina S punta ad un elemento del primo blocco e legge γ˙, allora la testina del primo nastro è in questa posizione e legge γ);

- usiamo un alfabeto esteso Г2 tale che γ˙ ∈ Г2 per ogni γ ∈ Г.