2. ESPRESSIONI REGOLARI

Una ***espressione regolare*** è un modo dichiarativo per descrivere un ***linguaggio regolare***. Possiamo usare le ***operazioni regolari*** per costruire espressioni che descrivono ***linguaggi***, che sono chiamate ***espressioni regolari***.

$$(0 \cup 1)0^*$$

Il valore di questa espressione è un linguaggio, che consiste di tutte le stringhe che iniziano con un 0 o un 1, seguito da qualsiasi numero di simboli uguali a 0. La prima parte, ovvero i simboli 0 e 1 sono le abbreviazioni per gli insiemi {0} e {1}, quindi (0 ∪ 1) significa ({0} ∪ {1}). Il valore di questa parte è il linguaggio {0, 1}. La seconda parte, ovvero 0* denota {0}*, il suo valore è il linguaggio che consiste di tutte le stringhe contenenti, cioè un numero qualsiasi di simboli uguali a 0 (anche nessuno).

Sia A = {0, 1}. Per brevità, nelle espressioni regolari: 0 indica {0}, 1 indica {1}. In pratica, eliminiamo la notazione insiemistica per brevità.

Es. 0 ∪ 1 indica {0} ∪ {1}, cioè {0, 1}.

Es. (0 ∪ 1)0* è {0, 1}{0}* (dove {0}*. = {ε, 0, 00, 000, …}.), cioè l'insieme di stringhe binarie che iniziano con 0 oppure 1 e continuano con degli 0 (anche nessuno).

Es. (0 ∪ 1)* è {0, 1}*, cioè l'insieme di tutte le stringhe su {0, 1}.

2.1 DEFINIZIONE FORMALE DI UNA ESPRESSIONE REGOLARE (*DEFINIZIONE INDUTTIVA*)

Le espressioni regolari possono essere definite formalmente utilizzando una definizione induttiva. La base è:
- a è espressione regolare per ogni a nell'alfabeto, e denota l'insieme {a};
- ε è espressione regolare, e denota l'insieme {ε};
- ∅ è espressione regolare, e denota l'insieme {∅}.

Siano, ora, R_1 e R_2 espressioni regolari che rappresentano i linguaggi L_1 e L_2. Si ha che:
- l'***unione*** ($R_1 \cup R_2$) rappresenta l'insieme $L_1 \cup L_2$;
- la ***concatenazione*** (R_1R_2) rappresenta l'insieme L_1L_2;
- la ***chiusura di Kleene*** $(R_1)^*$ rappresenta l'insieme L_1^*.

Ogni espressione regolare può essere costruita a partire dal risultato ottenuto.

<u>*Esempio*</u>:

Sia R_1 = 0 l'e.r. che rappresenta L_1 = {0}, e sia R_2 = 1 l'e.r. che rappresenta L_2 = {1}. Allora:
- l'unione delle due espressioni regolari ($R_1 \cup R_2$) = 0 ∪ 1 rappresenta l'insieme $L_1 \cup L_2$ = {0, 1};
- la chiusura di Kleene $(R_1)^*$ = 0* rappresenta l'insieme L_1^* = {0}*.

Siano E_1 = 0 ∪ 1 e E_2 = 0* espressioni regolari che rappresentano rispettivamente gli insiemi L_1' = {0, 1} e L_2' = {0}*; allora si ha che la concatenazione E_1E_2 = (0 ∪ 1)0* è un'espressione regolare che rappresenta il linguaggio $L_1'L_2'$ = {0, 1}{0}*.

> Se R è un'espressione regolare, allora L(R) denota il linguaggio ***generato*** da R.

<u>*Esempio*</u>**:**

R = (0 ∪ 1)0* è un'espressione regolare che genera il linguaggio L(R) = {0, 1}{0}*.

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

BASE:
- ε e ∅ sono espressioni regolari: L(ε) = {ε} e L(∅) = ∅;
- se a ∈ Σ, allora a è un'espressione regolare: L(a) = {a}.

PASSO: Se R_1 e R_2 sono espressioni regolari, allora:
- $R_1 \cup R_2$ è un'espressione regolare che rappresenta $L(R_1) \cup L(R_2)$;
- R_1R_2 è un'espressione regolare che rappresenta $L(R_1)L(R_2)$;
- R_1^* è un'espressione regolare che rappresenta $(L(R_1))^*$.

Le espressioni regolari possono essere create applicando le operazioni di unione, concatenazione e Kleene star. Possiamo utilizzare le parentesi per cambiare l'ordine usuale di precedenza per le operazioni regolari, che è il seguente: 1. ***Kleene star*** (*); 2. ***Concatenazione*** (·); 3. ***Unione*** (∪).

<u>*Esempi*</u>:

1. 01* ∪ 1 è raggruppato in (0(1*)) ∪ 1, cioè il linguaggio che riconosce la stringa 1 oppure le stringhe che iniziano con 0 e terminano con zero o più 1.
2. 00 ∪ 101* è il linguaggio formato da una stringa 00 o da stringhe inizianti con 10 seguite da zero o più 1.
3. Sia 0(0 ∪ 101)* un'espressione regolare. Prima si effettua la concatenazione di 101, poi si effettua l'unione tra 0 e 101, di quest'unione si effettua la Kleene star, ed infine si effettua la concatenazione. Allora:
   - 0101001010 appartiene al linguaggio;
   - 00101001 non appartiene al linguaggio;
   - 0000000 appartiene al linguaggio;
   - 101 non appartiene al linguaggio.
4. Sia A = {0, 1}. Allora:
   1. (0 ∪ 1) genera il linguaggio {0, 1};
   2. 0*10* genera il linguaggio {w | w ha un solo 1};
   3. A*1A* genera il linguaggio {w | w ha almeno un 1};
   4. A*001A* genera il linguaggio {w | w ha 001 come sottostringa};
   5. (AA)* genera il linguaggio {w | |w| è pari}, in quanto (AA)* = ({0, 1}{0, 1})* = {00, 01, 10, 11}*;
   6. (AAA)* genera il linguaggio {w | |w| è multiplo di 3}.
5. Sia EVEN-EVEN su A = {a, b} l'insieme di stringhe con un numero pari di a e un numero pari di b. Ad esempio, aababbaaababab ∈ EVEN-EVEN. Si ha che l'e.r. (aa ∪ bb ∪ (ab ∪ ba)(aa ∪ bb)*(ab ∪ ba))* genera il linguaggio EVEN-EVEN.
   Questo esempio fa comprendere che le espressioni regolari consentono di definire una classe di linguaggi che è esattamente quella dei linguaggi regolari.

2.2 EQUIVALENZA CON GLI AUTOMI

Ogni espressione regolare può essere trasformata in un automa finito che riconosce il linguaggio che essa descrive, e viceversa.

Teorema di Kleene:

Un linguaggio è ***regolare*** se e solo se qualche ***espressione regolare*** lo descrive: L ↔ E.R.

Questo teorema va dimostrato in entrambe le direzioni, dimostrandolo in due lemma separati.

Lemma 1:

Se un linguaggio è descritto da un'***espressione regolare***, allora esso è ***regolare***.

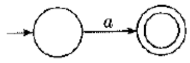
Idea:

Supponiamo di avere un'espressione regolare R che descrive un linguaggio A. Trasformiamo R in un NFA che riconosce A.

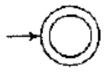
Dimostrazione:

Consideriamo i sei casi nella definizione di espressione regolare.

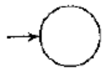
1. R = *a* per qualche *a* ∈ ∑, allora L(R) = {a} e il seguente NFA riconosce L(R).
   Nota: questa macchina soddisfa la definizione di NFA ma non quella di DFA, perché ha qualche stato con nessun arco uscente per ogni possibile simbolo di input. Con NFA è più facile da descrivere.



2. R = ε, allora L(R) = {ε} e il seguente NFA riconosce L(R).



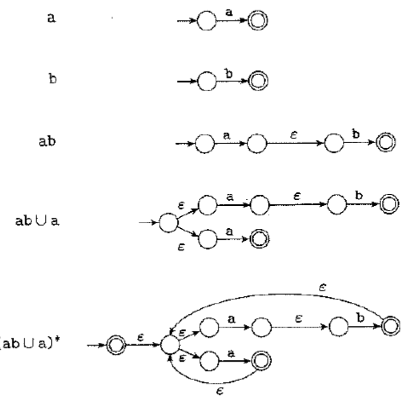
3. R = ∅, allora L(R) = ∅ e il seguente NFA riconosce L(R).



4. R = R_1 U R_2.
5. R = R_1 o R_2.
6. R = R_1^*.
   Per questi ultimi 3 casi si usano le costruzioni date nelle prove che la classe dei linguaggi regolari è chiusa rispetto alle operazioni regolari.

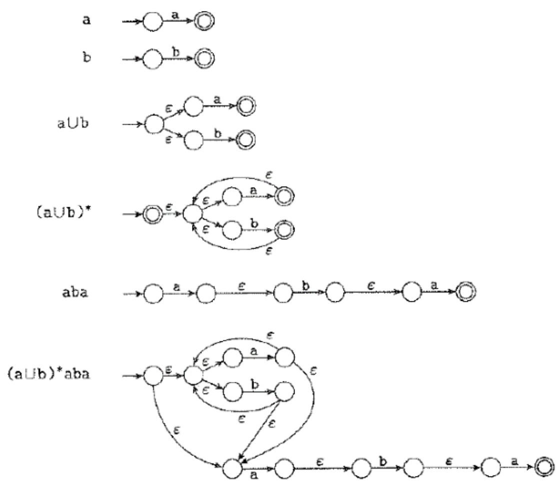
Esempio1:

Trasformiamo l'espressione regolare (ab U a)* in un NFA in una sequenza di passi. Costruiamo l'automa dalle sotto-espressioni più piccole alle sotto-espressioni più grandi, finché non abbiamo un NFA per l'espressione iniziale.



Esempio2:

Trasformiamo l'espressione regolare (a U b)*aba in un NFA.



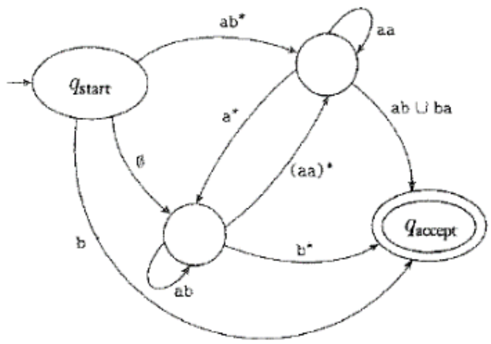
Lemma 2:

Se un linguaggio è ***regolare***, allora è descritto da un'***espressione regolare***.

Idea:

Dobbiamo mostrare che il linguaggio A è regolare, allora un'espressione regolare lo descrive. Poiché A è regolare, esso è accettato da un DFA.

Per trasformare un DFA in una espressione regolare equivalente, si divide la procedura in due parti, usando un nuovo tipo di ***automa*** quello ***finito non deterministico generalizzato, GNFA***, ovvero un NFA che può avere archi delle transizioni con espressioni regolari come etichette, invece che solo elementi dell'alfabeto o ε.



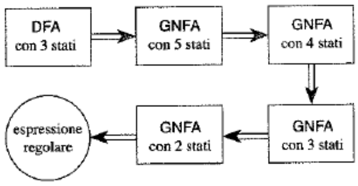
Per comodità i GNFA devono soddisfare le seguenti condizioni:

- Lo stato iniziale ha archi di transizione uscenti verso un qualsiasi altro stato ma nessun arco entrante proveniente da un qualsiasi altro stato.
- Esiste un solo stato accettante, ed esso ha archi entranti provenienti da un qualsiasi altro stato, ma nessun arco uscente verso un qualsiasi altro stato. Inoltre, lo stato accettante non è uguale allo stato iniziale.
- Eccetto che per lo stato iniziale e lo stato accettante, un arco va da ogni stato ad ogni altro stato e anche da ogni stato in sé stesso.

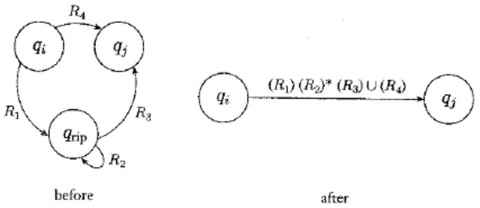
Possiamo facilmente trasformare un DFA in un GNFA nella forma speciale. Aggiungiamo semplicemente un nuovo stato iniziale con un ε-arco che entra nel vecchio stato iniziale e un nuovo stato accettante con ε-archi entranti, provenienti dai vecchi stati accettanti.

Per trasformare un ***GNFA*** in un'***espressione regolare***, supponiamo che GNFA abbia k stati. Poiché un GNFA deve avere uno stato iniziale e uno accettante ed essi devono essere diversi tra loro, sappiamo che k è maggiore o uguale a 2. Se k è maggiore di 2, costruiamo un GNFA equivalente con k-1 stati. Questo passo viene ripetuto sul nuovo GNFA fino a quando esso è ridotto a due stati, e l'etichetta dallo stato iniziale al finale rappresenta l'espressione regolare equivalente.

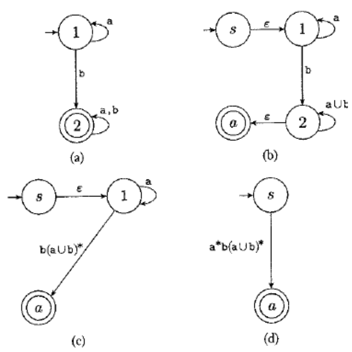
Il passo cruciale è costruire un GNFA equivalente con uno stato in meno quando k è maggiore di 2. Lo facciamo scegliendo uno stato, estraendo dalla macchina, e modificando il resto in modo che sia ancora riconosciuto lo stesso linguaggio. Ogni stato può essere scelto, purché non sia lo stato iniziale o quello accettante.

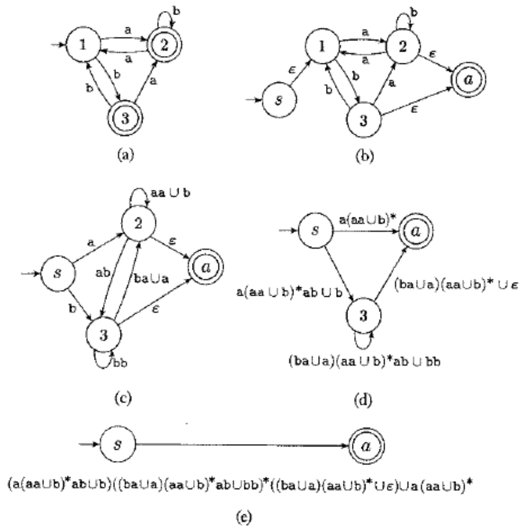


Esempio:
Assumiamo che q_{rip} sia lo stato rimosso, dopo averlo rimosso modifichiamo la macchina cambiando le espressioni regolari che etichettano ciascuno dei restanti archi. Le nuove etichette controbilanciano l'assenza di q_{rip} reintegrando le computazioni perse. La nuova etichetta che va da uno stato q_i ad uno stato q_j è una espressione regolare che descrive tutte le stringhe che porterebbero la macchina da q_i a q_j o direttamente o tramite q_{rip}.



Altri esempi:





2.3 LINGUAGGI NON REGOLARI

Esistono alcuni linguaggi che non possono essere riconosciuti da alcun automa finito. Consideriamo il linguaggio $B=\{0^n1^n \mid n\geq 0\}$, se si cerca di trovare un DFA che riconosca B, si osserva che la macchina ha bisogno di ricordare quanti simboli uguali a 0 ha visto fin quando legge input. Poiché il numero di simboli uguali a 0 non è limitato, la macchina dovrà tenere traccia di un numero infinito di possibilità. Ma non può farlo con un numero finito di stati.

Esempio:

Il linguaggio $L = \{a^nb^n \mid n \geq 0\}$ non è regolare, cioè non è riconosciuto da alcun automa finito: non è possibile, infatti, avere un automa che riconosca un linguaggio di questo tipo. Se tentiamo di costruire un DFA che riconosce $L = \{a^nb^n \mid n \geq 0\}$, l'automa deve verificare che la stringa è fatta da un numero *n* di a seguito da uno stesso numero *n* di b: ci servirebbero infiniti stati per contare il numero di a ed il numero di b.

2.3.1 PUMPING LEMMA

Per provare la ***non regolarità*** si usa la tecnica che deriva dai linguaggi regolari, chiamato ***pumping lemma***. Questo teorema afferma che tutti i linguaggi regolari hanno una proprietà speciale, se si mostra che un certo linguaggio non soddisfa tale proprietà, si ha la certezza che esso non è regolare.

La proprietà afferma che tutte le stringhe nel linguaggio possono essere "replicate" se la loro lunghezza raggiunge almeno uno specifico valore speciale, chiamato la "***lunghezza del pumping***". Questo significa che ogni tale stringa contiene una parte che può essere ripetuta un numero qualsiasi di volte ottenendo una stringa che appartiene ancora al linguaggio.

TEOREMA 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*** $i \geq 0$, $xy^iz \in A$;
2. $|y| > 0$;
3. $|xy| \leq p$.

Nota: Ricordiamo che $|s|$ rappresenta la lunghezza della stringa s, y^i indica i copie di y concatenate insieme, e y^0 è uguale a ε.

Dimostrazione:

Siano M un automa che riconosce L, e p il numero di stati di M.

Considerando una stringa w = xyz tale che $|w| \geq p$, allora sappiamo che nella computazione esiste (almeno) uno stato ripetuto, cioè esiste (almeno) uno stato che viene visitato più di una volta (sia r il primo stato ripetuto). A questo punto, sappiamo che:

- x (la prima parte della stringa w) porta la computazione dallo stato iniziale q_1 allo stato r (cioè, $f^*(q_1, x) = r$);
- y (la seconda parte della stringa w) porta la computazione dallo stato r allo stato r (cioè, $f^*(r, y) = r$);
- z (la terza parte della stringa w) porta la computazione dallo stato r allo stato finale (cioè, $f^*(r, z) \in F$).

Verifichiamo che questa suddivisione soddisfa le tre proprietà del lemma:

- $|xy| \leq p$ (con r primo stato ripetuto) è soddisfatta, in quanto se così non fosse allora avremmo lo stato ripetuto prima;
- $|y| \geq 1$ segue dalla precedente, in quanto abbiamo bisogno di almeno un simbolo per poter ciclare su r;
- xy^iz appartiene al linguaggio L, dato che tale stringa porta dallo stato iniziale ad r, *da r ad r per i volte* (anche 0 volte), e infine da r a uno stato finale.

TEOREMA:

Sia L l'insieme di tutte le stringhe 0^n1^n (con $n \geq 0$) su {0, 1} aventi un uguale numero di 0 e di 1. Il linguaggio L ***non è regolare***.

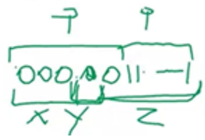
Dimostrazione:

Dimostriamo questo teorema per contraddizione usando il Pumping Lemma (PL). Supponiamo, a tal scopo, che L è regolare, quindi applichiamo il lemma. Sia, quindi, p la costante del PL. Consideriamo una stringa $w = 0^p1^p$. Il pumping lemma implica che esistono $xyz = 0^p1^p$, tali che $|xy| \leq p$, y è non vuota e che xy^kz appartiene a L *per ogni* $k \geq 0$.

Ma se la stringa presa è 0^p1^p, allora i primi p simboli sono tutti 0. Tuttavia, se il PL dice che la lunghezza della sottostringa xy è effettivamente minore o uguale a p, allora sappiamo che essa sta entro quei p simboli 0; inoltre, siccome y è non vuota, abbiamo che y è una sottostringa di almeno uno 0.

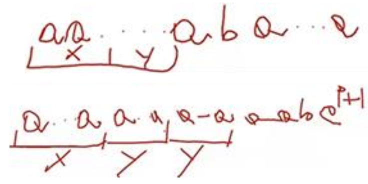
Ora, consideriamo la stringa xy^kz. Presa in input tale stringa, per contraddire il PL occorre esibire un valore k per cui la stringa xy^kz non appartiene al linguaggio.

Consideriamo $k = 2$ (quindi, y si ripete due volte). Per appartenere al linguaggio, questa stringa dovrà avere un numero p di 0 ed un numero p di 1: abbiamo una stringa del tipo $0^{|x|}0^{|y|}0^{|y|}0 \cdots 01^p = 0^{(p + |y|)}1^p$; poiché $|y| > 1$, si ha che questa stringa non appartiene al linguaggio L, dato che $p + |y| > p$.



Ricapitolando, preso un qualsiasi $k > 1$, allora xy^kz e xyz hanno (tra di loro) diverso numero di 0 e stesso numero di 1. Questa è una ***contraddizione***.

Esempio:
Sia $L = \{a^nba^m \mid n < m\}$. Usiamo il Pumping Lemma per dimostrare che il linguaggio L non è regolare.
Supponiamo P.A. che L sia regolare; di conseguenza, esso soddisferebbe il PL, quindi esisterebbe una costante p tale che, per ogni stringa w in L di lunghezza $|w| \geq p$, esistono stringhe x, y, z tali che $w = xyz$, che soddisfano: $|y| > 0$, $|xy| \leq p$, xy^iz in L (per ogni $i \geq 0$).
Sia definita la stringa, legata a p, $w = a^pba^{p+1} = xyz$, dove $|xy| \leq p$; da questo consegue che xy è costituita da un numero finito di a.
Bisogna dimostrare che la terza condizione non è soddisfatta, cioè che esiste $i \geq 0$ tale che la stringa $xy^iz \notin L$.
Sia, ad esempio, $i = 2$, quindi xy^2z è la stringa a lato.



Notiamo che questa stringa non appartiene al linguaggio, dato che ($|x| + |y| +$ # a prima del simbolo b) = p, e quindi il numero totale di a prima del simbolo b è $p + |y|$; siccome $|y| > 1$, abbiamo che $p + |y| \geq p + 1$, quindi la condizione $n < m$ del linguaggio non è soddisfatta. Dal Pumping Lemma, otteniamo che il linguaggio L non è regolare. Questa è una contraddizione.