Espressioni regolari
descrivono i linguaggi regolari

- Un FA (NFA o DFA) è un metodo per costruire una macchina che riconosce linguaggi regolari.

- **Una espressione regolare e' un modo dichiarativo per descrivere un linguaggio regolare.**

- Esempio: $01^* \cup 10^*$

- Le espressioni regolari sono usate, ad esempio, in comandi UNIX (grep) strumenti per l'analisi lessicale di UNIX (Lex (Lexical analyzer generator) e Flex (Fast Lex)).

Espressioni regolari
descrivono i linguaggi regolari

- Un FA (NFA o DFA) è un metodo per costruire una macchina che riconosce linguaggi regolari.

- **Una espressione regolare e' un modo dichiarativo per descrivere un linguaggio regolare.**

- Esempio: $01^* \cup 10^*$

- Le espressioni regolari sono usate, ad esempio, in comandi UNIX (grep) strumenti per l'analisi lessicale di UNIX (Lex (Lexical analyzer generator) e Flex (Fast Lex)).

Espressioni regolari

Sia $A=\{0, 1\}$. Per brevità nelle espressioni regolari:

- 0 indica {0},
- 1 indica {1}

Es: 0 U 1 indica {0}U{1}, cioè {0, 1}.

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

Espressioni regolari: definizione induttiva

BASE:

- a è espressione regolare per ogni a nell'alfabeto; denota l'insieme {a}
- ε è espressione regolare; denota l'insieme {ε}
- ϕ è espressione regolare; denota l'insieme ϕ

Espressioni regolari: definizione induttiva

Siano R1 e R2 espressioni regolari che rappresentano i linguaggi L1 e L2.

- UNIONE:

  (R1 U R2) rappresenta l'insieme L1 U L2.

- CONCATENAZIONE:

  (R1 R2) rappresenta l'insieme L1 L2.

- CHIUSURA DI KLINE:

  (R1)* rappresenta l'insieme L1*.

Linguaggio generato da espressioni regolari

Def: se R è espressione regolare, allora L(R) denota il linguaggio **generato** da R.

Definizione Induttiva di espressione regolare (e.r.)

Base:

- ε e $\emptyset$ sono espressioni regolari: $L(\varepsilon) = \{\varepsilon\}$ e $L(\emptyset) = \emptyset$.
- Se *a* in Σ, allora *a* è un'espressione regolare: $L(a) = \{a\}$.

Passo:
Se R1 e R2 sono e.r., allora

- R1 U R2 è e.r. che rappresenta L(R1) U L(R2).
- R1R2 è un' e.r. che rappresenta L(R1)L(R2).
- $R1^*$ è un' e.r. che rappresenta $(L(R1))^*$.

Nota.

Definizione induttiva suggerisce modo generale di provare generico teorema T per ogni e.r.

Passo 1: *T* vero per casi base

Passo 2: i.i. *T* corretto per R_1 e R_2
Mostriamo che T vero per
$(R_1 \cup R_2)$, R_1R_2, $(R_1)^*$

Gerarchia di operazioni in espressioni regolari

In aritmetica, moltiplicazione ha precedenza su addizione.

$2+3 \times 4 = 14$.

Parentesi cambiano ordine usuale: $(2+3) \times 4 = 20$.

Ordine di precedenza per operazioni regolari:

1. Kleene star
2. Concatenazione
3. Unione

Parentesi cambiano ordine usuale

Gerarchia di operazioni in espressioni regolari

Ordine di precedenza per operazioni regolari

1. Kleene star (*)
2. Concatenazione (.)
3. Unione (U)

Es.: 01^* U 1 è raggruppato in $(0(1)^*)$ U 1

Es: 00 U 101* linguaggio formato da stringa 00 e da stringhe inizianti con 10 seguita da zero o più 1.

Gerarchia di operazioni in espressioni regolari

Ordine di precedenza per operazioni regolari

1. Kleene star (*)
2. Concatenazione ()
3. Unione (+)

Es: 0(0 U 101)*

- **0101001010 in linguaggio?**
- **00101001?**
- **0000000?**
- **101?**

Esempi di espressioni regolari

Es: $A=\{0, 1\}$,

1. $(0 \cup 1)$ linguaggio $\{0, 1\}$

2. 0^*10^* linguaggio $\{w \mid w$ ha un solo 1 $\}$

3. A^*1A^* linguaggio $\{w \mid w$ almeno un 1 $\}$

4. A^*001A^* linguaggio $\{w \mid w$ ha 001 come sottostringa $\}$

5. $(AA)^*$ linguaggio $\{w \mid |w|$ pari $\}$

6. $(AAA)^*$ linguaggio $\{w \mid |w|$ multiplo di 3 $\}$

Es:
Sia EVEN-EVEN su A={a, b} insieme stringhe con numero pari di a e numero pari di b

Es, aababbaaababab in EVEN-EVEN.

espressione regolare:

$$(aa \cup bb \cup (ab \cup ba)(aa \cup bb)^*(ab \cup ba))^*$$

Teorema di Kleene

Teorema Linguaggio L è regolare sse L ammette una espressione regolare.

IDEA DIM. Teorema di Kleene

Sappiamo

Linguaggio L è regolare ⇔ L riconosciuto da NFA

⇔ L riconosciuto da DFA

Si mostra

Per ogni DFA A possiamo costruire un'e.r. R con L(R) = L(A).
Per ogni e.r. R esiste un NFA A, tale che L(A) = L(R).
Cioè
L riconosciuto da DFA ⇔ L ammette un' e.r.

Quindi

Linguaggio L è regolare ⇔ L ammette un' e.r.

L ammette un' e.r.→ L riconociuto da NFA

1. Dimostriamo per casi base: a, e, f
2. Assumiamo correttezza per R1 e R2, proviamo per R1 U R2, R1R2, R1* .

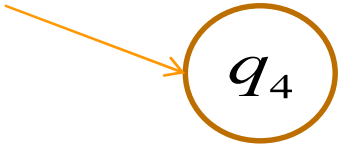
L ammette un' e.r.→ L riconociuto da NFA

1. Dimostriamo per casi base: a, ε, ϕ.

- a:

- ε:

- ϕ;



L ammette un' e.r.→ L riconociuto da NFA

1. Dimostriamo per casi base: a, e, f
2. Assumiamo correttezza per R1 e R2, proviamo per R1 U R2, R1R2, R1*

Immediato: sappiamo costruire NFA per Unione, concatenazione e Klene star.

L ammette un' e.r.➔ L riconociuto da NFA

1.Dimostriamo per casi base: a, e, f
2.Assumiamo correttezza per R1 e R2, proviamo
per R1 U R2, R1R2, R1*

Es. Costruire NFA per

$$(ab \cup a)^*$$

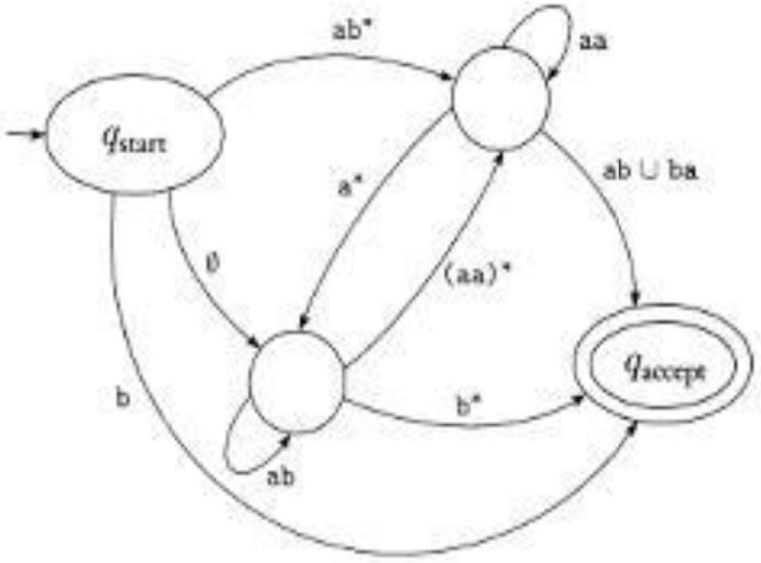
$$(a \cup b)^* aba$$

L riconociuto da DFA ➔L ammette un' e.r.

Procedura di conversione dei DFA in espressioni regolari equivalenti.

Costruiamo prima variazione di NFA:

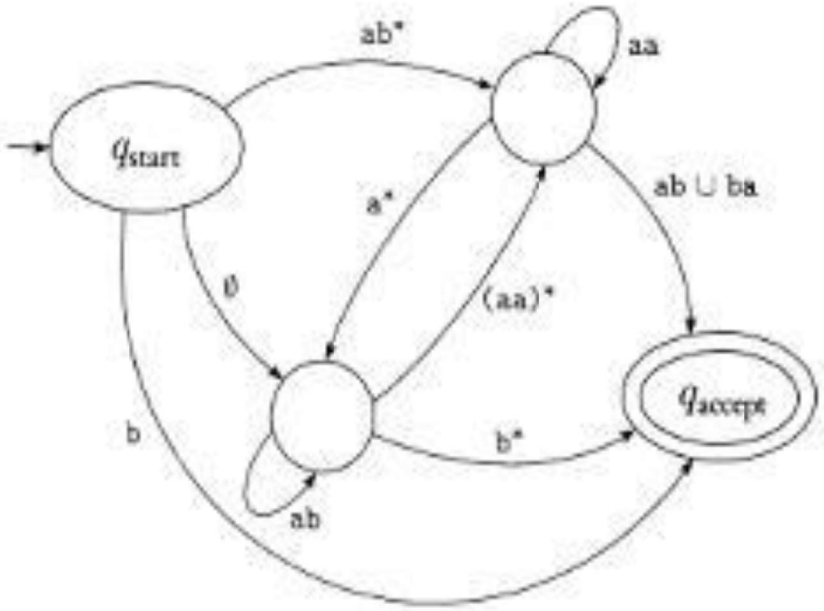
le transizione possono avere delle espressioni regolari per etichette, invece che solo gli elementi dell'alfabeto o ε



L riconociuto da DFA ➔L ammette un' e.r.

Per comodità vogliamo:

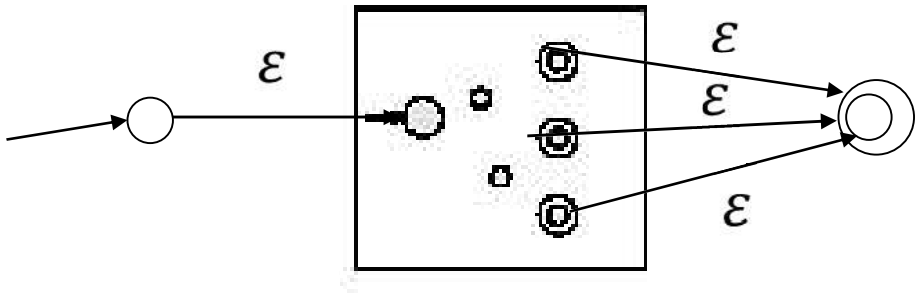
- Lo stato iniziale non ha transizioni da qualsiasi altro stato.
- C'e soltanto un singolo stato-accetta; non ha frecce uscenti e non coincide con lo stato-start.



L riconociuto da DFA ➔L ammette un' e.r.

Per convertire un DFA

1. aggiungere un nuovo stato-start collegato con una ε-transition al vecchio stato start
2. e un nuovo stato-accetta collegato con ε-transitions dai vecchi stati-accetta.



L riconociuto da DFA ➔L ammette un' e.r.

3. Costruzione di macchina equivalente con meno stati quando il numero di stati è k> 2:

•Selezioniamo uno stato, lo eliminiamo dalla macchina e sistemiamo il resto in modo che sia ancora riconosciuto lo stesso linguaggio.

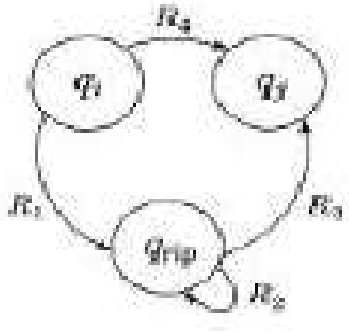
Qualsiasi stato va bene, a condizione che non sia start o accetta.

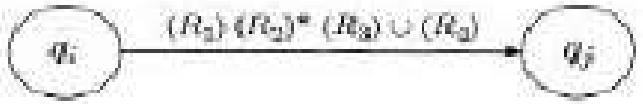
Abbiamo la garanzia che tale stato esiste perché k> 2.

- Chiamiamo lo stato rimosso q_{rip}.
- Dopo aver rimosso q_{rip} modifichiamo la macchina alterando le espressioni regolari che etichettano le restanti frecce.

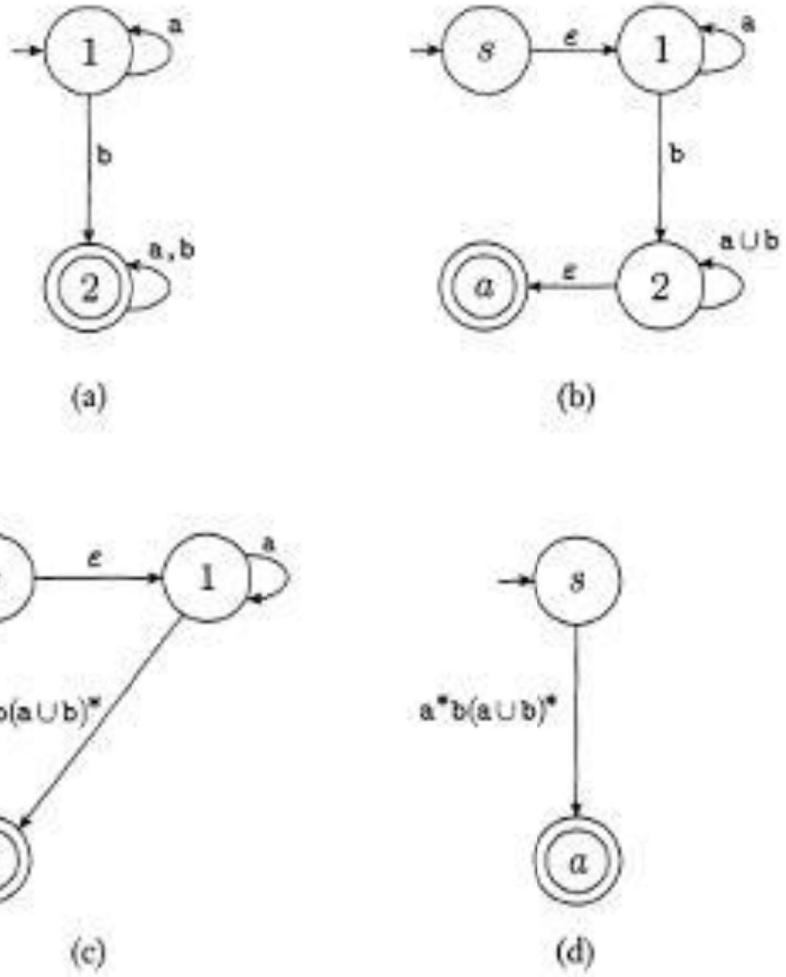
  Le nuove etichette compensano l'assenza di q_{rip} aggiungendo le computazioni perdute nella modifica:

  La nuova etichetta passando da uno stato qi ad uno stato qj è una espressione regolare che descrive tutte le stringhe che avrebbero portato la macchina da qi a qj direttamente o attraverso q_{rip}





L riconociuto da DFA ➔L ammette un' e.r.



Nota: Noi diamo il concetto formale di base, la pratica verrà ….

Google: Searches related to regular expressions use

how to use regular expressions **in python**
how to use regular expressions **in excel**
how to use regular expressions **in perl**
how to use regular expressions **in ruby**
how to use regular expressions **in notepad++**
how to use regular expressions **in java**
how to use regular expressions **in sublime text**
how to use regular expressions **in javascript**

programmers.stackexchange.com a question and answer site for professional programmers interested in conceptual questions about software development.

Question: Is it a must for every programmer to learn regular expressions?
I am new to programming, and at an interview I got a question on regular expressions; needless to say I couldn't answer. So I was wondering whether I should learn regular expression? Is it a must for every programmer of all fields? Or it is a must for programming for some particular fields?
Answers: Regular expressions are such an incredibly convenient tool, available across so many languages that most developers will learn them sooner or later. For an interviewer, they are a nice way to probe experience during an interview. If you are interviewing someone claiming years of experience who doesn't understand them, you need to dig further

Tutti i linguaggi sono regolari?

La risposta è negativa

Esempio:

$$L = \{a^n b^n \mid n \geq 0\}$$

?

Tutti i linguaggi sono regolari?

Se tentiamo di costruire DFA che riconosce

$$L = \{a^n b^n \mid n \geq 0\}$$

Ci accorgiamo che ci servono infiniti stati per "ricordare" il numero di a visti

(poi dovremo controllare che che numero di b coincide con quello di a)

Nota. Esiste dimostrazione formale

Pumping Lemma

Applicazione primaria: mostrare che un linguaggio non è regolare

Lemma: Per ogni linguaggio regolare L, esiste una costante positiva p tale che per ogni stringa w con in L di lunghezza $|w| \geq p$

Esistono stringhe x, y, z t.c. w = xyz che soddisfano:

- $|y| > 0$
- $|xy| \leq p$
- xy^iz in L, per ogni $i \geq 0$.

Dim. Pumping Lemma (Idea)

Per ogni linguaggio regolare L, esiste una costante positiva p tale che per ogni stringa w con in L di lunghezza $|w| \geq p$ esistono stringhe x, y, z, t.c. w = xyz, che soddisfano: $|y|>0$, $|xy| \leq p$, xy^iz in L, per ogni $i \geq 0$.

Siano: M automa che riconosce L, p=numero stati di M

$|w| > p$ ➔ nella computazione esiste stato ripetuto

(sia r il primo stato ripetuto)

➔ x porta da stato iniziale ad r,

y da r ad r,

z da r a stato finale

➔ $|xy| \leq p$ (r primo stato ripetuto), $|y| \geq 1$,

xy^iz porta da stato iniziale ad r, **da r ad r per i volte**,

da r a stato finale

Applicazione del Pumping Lemma

Teorema: L = insieme di tutte le stringhe su {0, 1} aventi un ugule numero di 0 e di 1. Il linguaggio L non è regolare

Dim: Per contraddizione usando il PL. Supponiamo L regolare.

Sia p la costante del PL.

Consideriamo stringa $w = 0^p1^p$

PL implica che esistono $xyz = 0^p1^p$, tali che $|xy| \leq p$,

y non vuota, e **xy^kz in L per ogni k > 0.**

➔

xy formata da tutti 0 (perchè?) ed y stringa di almeno uno 0.

Applicazione del Pumping Lemma

cont.

se k >1, allora xy^kz e xyz hanno (tra di loro):

- diverso numero di 0
- stesso numero di 1,

➔ In xy^kz il numero di 0 differisce da quello di 1

➔ xy^kz non in L, CONTRADDIZIONE!

➔ l'assunzione che L è regolare deve essere falsa

➔ L NON regolare

Applicazione del Pumping Lemma

Corollario: Il linguaggio $\{0^i1^i \mid i \geq 0\}$ non è regolare.

La dimostrazione è essenzialmente uguale alla precedente

Linguaggi regolari

- Linguaggi riconoscuti da automa finito deterministico
- Linguaggi riconoscuti da automa finito non deterministico
- Linguaggi generati da espressione regolare

Automi finiti

- Reti sequenziali ⇔ realizzazione fisica di automi finiti (con output)
- Compilatori: realizzazione del parser, analisi lessicale
- Software per la progettazione di circuiti digitali
- Strumenti per la specifica e la verifica di sistemi a stati finiti (sistemi biologici, protocolli di comunicazione)
- Realizzazione di strumenti di elaborazione testuale
- Ricerca di parole chiave in un file o sul web.

Esistono linguaggi non regolari