Indice

1 Operazioni sui linguaggi

Visto che un linguaggio $L \subseteq \Sigma^*$ è un sottoinsieme di un insieme, possiamo fare tutte le operazioni insiemistiche su un linguaggio:

- intersezione $L \cap L'$
- unione $L \cup L'$
- complemento L^C. Si può pensare di calcolare il complemento di un linguaggio $L \subseteq \Sigma^*$ rispetto ad un alfabeto più grande $\Sigma \subseteq \Gamma$. Il complemento è esattamente $\Gamma^* \setminus L$.

Definiamo il prodotto di due linguaggi $L', L'' \subseteq \Sigma^*$

$$L' \cdot L'' = \{z \mid \exists x \in L' \exists y \in L''.z = xy\}$$

E se $L' \subseteq \Sigma^{*\prime}, L'' \subseteq \Sigma^{*\prime\prime}$, allora $L' \cdot L'' \subseteq \Sigma' \cup \Sigma''$. Salvo casi particolari (e.g. alfabeto da una lettera sola) questa operazione non è commutativa. E vale $\{\epsilon\}$ è l'indentità destra e sinistra, e $\varnothing$ è l'elemento nullo destro e sinistro.

Definiamo la potenza di un linguaggio

$$L^k = \underbrace{L \cdot L \dots}_{k \text{ volte}}$$

e vale che

$$L^0 = \{\epsilon\}$$

ovvero il linguaggio ottenuto concatenando zero parole di L. Il prodotto e la potenza di un linguaggio finito sono ancora finiti.

Definiamo la chiusura di Kleene di un linguaggio

$$L^* = \bigcup_{k \geq 0} L^k$$

La chiusura di Kleene anche di un linguaggio finito è finito.

$$\begin{aligned} L &= \{a, bb\}^* \\ &= \{x \in \{a, b\}^* \mid \text{ Ogni fattore massimale di } b \text{ è di lunghezza pari } \} \end{aligned}$$

Dove massimale indica che non può essere allungato.

Vale inoltre che

$$\{\epsilon\}^* = \{\epsilon\}$$

e che

$$\varnothing^* = \{\epsilon\}$$

Definiamo la chiusura di Kleene positiva di un linguaggio

$$L^+ = \bigcup_{k \geq 1} L^k$$

Inoltre vale che L non contiene la parola vuota, allora L^+ a sua volta non la contiene, e questo è uguale a $L^+ = L^* \setminus \{\epsilon\}$. Nota bene, non vale che generalmente $L^+ = L^* \setminus \{\epsilon\}$.

Fatto 1.

$$L \cdot L^* = L \bigcup_{k \geq 0} L^k = \bigcup_{k \geq 0} LL^k = \bigcup_{k \geq 1} L^k = L^+ = L^* \cdot L$$

2 Espressioni regolari

Le espressioni regolari sono in un certo senso una descrizione dichiarativa di un linguaggio. Dato un alfabeto Σ, definiamo ricorsivamente le espressioni regolari come

- $\varnothing$, rappresenta il linguaggio vuoto
- ϵ, rappresenta il linguaggio della parola vuota
- $a \in \Sigma$, rappresenta il linguaggio di solo a

ora definiamo

- $E_1 + E_2$, rappresenta il linguaggio $L(E_1) \cup L(E_2)$
- $E_1 \cdot E_2$, rappresenta il linguaggio $L(E_1) \cdot L(E_2)$
- E^*, rappresenta il linguaggio $L(E)^*$

L'espressione regolare

$$(a + bb)^*$$

rappresenta il linguaggio $\{a, bb\}^*$.

Il linguaggio dove il terzultimo simbolo è una a è rappresentato dall'espressione regolare

$$(a + b)^* a(a + b)(a + b)$$

E volendo generalizzare al linguaggio il cui n-esimo simbolo da destra è una a

$$(a + b)^* a(a + b)^n$$

Dove le potenze sulle espressioni regolari rappresentano una serie di prodotti.

Il linguaggio dove due simboli a distanza n sono uguali

$$((a + b)^* a(a + b)^{n-1} a(a + b)^*) + ((a + b)^* b(a + b)^{n-1} b(a + b)^*)$$

Questo può essere semplificato in

$$(a + b)^*((a(a + b)^{n-1} a) + (b(a + b)^{n-1} b))(a + b)^*$$

Definizione 1 (State complexity)**.** *Definiamo la complessità di stati o state complexity di un linguaggio* $L \subseteq \Sigma^*$*, indicata* $\mathrm{sc}(L)$*, come il minimo numero di stati del DFA che accetta* L*. E definiamo la non-deterministic state complexity, indicata con* $\mathrm{nsc}(L)$*, come il minimo numero di stati del NFA che accetta* L*.*

Fatto 2.

$$\mathrm{sc}(L) \leq 2^{\mathrm{nsc}(L)}$$

Dato $L_n = (a+b)^*a(a+b)^{n-1}$, vale che $\mathrm{nsc}(L_n) \leq n+1$, mentre $\mathrm{sc}(L_n) \geq 2^n$, e visto che per questo linguaggio sappiamo costruire un automa da 2^n stati, allora $\mathrm{sc}(L_n) = 2^n$.
La stringa più corta di questa linguaggio è la stringa che inizia con a ed ha n simboli. Supponendo di avere meno di n stati, allora ce ne deve essere almeno uno che si ripete; quindi c'è un loop. Se io elimino il loop potrei riconoscere una stringa più breve di quella più breve.
Quindi abbiamo che per forza $\mathrm{nsc}(L_n) = n+1$.
Alternativamente si può costruire il fooling set composto da tutte le possibili scomposizioni in due stringhe di ab^{n-1}.

Proposizione 1. *Presa la stringa più corta del linguaggio, ogni NFA deve per forza avere $n+1$ stati, dove n sono i simboli della stringa.*

Teorema 1 (Teorema di Kleene o Teorema fondamentale degli automi a stati finiti)**.** *La classe dei linguaggi accettati da automi a stati finiti è la più piccola sottoinsieme di Σ^* che contiene i linguaggi finiti ed è chiusa rispetto alle operazioni di unione, prodotto e chiusura di Kleene.*

Quindi la classe degli automi a stati finiti coincide con la classe esprimibile dalle espressioni regolari.

Dimostrazione. Dimostriamo il primo lato, passandro da un automa ad una regex. Dato un automa

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_1, F \rangle$$

e supponiamo che gli stati siano numerati da uno ad n

$$Q = \{q_1, q_2, \ldots, q_n\}$$

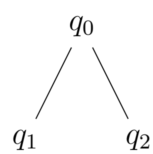
possiamo vedere come il linguaggio come tutti i cammini dallo stato iniziale agli stati finali, e che concantenando le etichette del cammino otteniamo un parola accettata dal percorso.le etichette del cammino otteniamo un parola accettata dal percorso.

Costruiamo un algoritmo simile a quello di Floyd-Wharshall che trova tutti i camminini minimi. Chiamiamo $R_{ij}^{(k)}$ l'insieme di tutti i percorsi da q_i a q_j che passano solo per stati con indice minore di k. Quando metteremo $k = n$ avrò trovato tutte le stringhe da i a j. Andando per induzione definiamo l'insieme come

- $R_{ij}^{(0)}$ è il caso in cui c'è una transizione tra lo stato i e lo stato j.

$$R_{ij}^{(0)} = \begin{cases} \{\alpha \in \Sigma \mid \delta(q_i, \alpha) = q_j\} & \text{se } i \neq j \\ \{\epsilon\} \cup \{\alpha \in \Sigma \mid \delta(q_i, \alpha) = q_i\} & \text{se } i = j \end{cases}$$

- supponiamo di avere $R_{ij}^{(k-1)}$, voglio trovare $R_{ij}^{(k)}$. Cerco tutti i punti per cui il nuovo cammino passa per q_k, nelle sezioni tra i q_k gli stati intermedi sono tutti minori di k.



Quindi

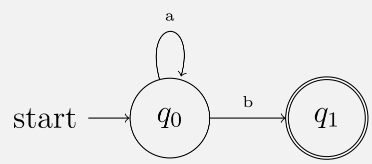
$$R_{ij}^{(k)} = R_{ij}^{(k-1)} \cup R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

Con $k = n$ abbiamo tutti i percorsi e vale che

$$L = \bigcup_{q_i \in F} R_{1i}^{(n)}$$

Dimostriamo il secondo lato, passandro da una regex ad un automa. ■

Mostriamo una tecnica diversa per generare la regex da un automa.



Costruiamo un sistema di equazioni per ogni stato

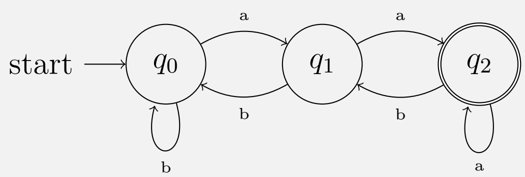
$$\begin{cases} X &= aX_1 + bY \\ Y &= \epsilon \end{cases}$$

e sostituendo

$$\begin{cases} X &= aX + b \\ Y &= \epsilon \end{cases}$$

se ci troviamo in uno stato $X = AX + B$, questo corrisponde alla regex A^*B. Questo è banalmente a^*b.

Un altro esempio più complesso



Costruiamo un sistema di equazioni per ogni stato

$$\begin{cases} X_0 & = aX_1 + bX_0 \\ X_1 & = aX_2 + bX_0 \\ X_2 & = aX_2 + bX_1 + \epsilon \end{cases}$$

ϵ perché X_2 è finale. Sostitendo X_1 abbiamo

$$\begin{aligned} X_0 &= aaX_2 + abX_0 + bX_0 \\ &= aaX_2 + (ab + b)X_0 \end{aligned}$$

e

$$\begin{aligned} X_2 &= aX_2 + baX_2 + bbX_0 + \epsilon \\ &= (a + ba)X_2 + bbX_0 + \epsilon \\ &= (a + ba)^* + bbX_0 + \epsilon \end{aligned}$$

che corrisponde alla regex $(a + ba)^*(bbX_0 + \epsilon)$, prendendo come $A = (a + ba)$ e $B = (bbX_0 + \epsilon)$. E sostituendo ancora

$$\begin{aligned} X_0 &= aaX_2 + (ab + b)X_= \\ &= aa(a + ba)^*(bbX_0 + \epsilon) + (ab + b)X_0 \\ &= (aa(a + ba)^*bb + ab + b)X_0 + aa(a + ba)^* \end{aligned}$$

che quindi è $(aa(a + ba)^*bb + ab + b)^* + aa(a + ba)^*$.

Il complemento di linguaggio riconosciuto da un automa deterministico è banalmente il complemento dell'insieme dei finali.