UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Elementi e teoria della computazione

"La sacra Bibbia di ETC"

Relatore:
Ch.mo Prof.
Luisa Gargano

Correlatore:
Ch.mo Prof.
Gianluca De Marco

Candidati:
Luca Boffa
Mat. 051216052
Matteo Della Rocca
Mat. 0512105830

ANNO ACCADEMICO 2021/2022

Dedicata a loro che
ci provano da tanti anni...

Indice

Chapter 0

Introduzione

0.1 Introduzione

Quest'opera è stata scritta con l'intento di facilitare lo studio e la comprensione del corso di "Elementi e teoria della computazione" del corso di informatica dell'Università di Salerno.
Il materiale qui scritto deriva in gran parte dagli appunti presi da me a lezione e qualche approfondimento dal libro.
Gli appunti in questione fanno fede al materiale didattico proposto a lezione nell'anno accademico 2021/2022(3^{nd} anno resto 1) dalla Professoressa L. Gargano e il Professore. G.De Marco.

0.1.1 Alcuni richiami utili

Chapter 1

Linguaggi regolari

La teoria della computazione ci fa sorgere una domanda:

"**Che cos'è una computer?**"

A primo impatto può sembrare una domanda sciocca poiché tutti ormai usano i computer quotidianamente. Ma i computer reali sono piuttosto complessi per sviluppare una **teoria matematica** direttamente su di essi. Perciò usiamo un computer ideale, chiamato **modelle di computazione**. Useremo diversi modelli di computazione a seconda delle caratteristiche richieste.

Iniziamo con i più semplici, ovvero le **macchine a stati finiti** o anche chiamati **automi finiti**.

1.1 Automi Finiti

Un **automa finito** o **DFA** è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove:

1. Q è un insieme finito degli **stati**
2. Σ è un insieme finito chiamato **alfabeto**
3. $\delta : Q \times \Sigma \longrightarrow Q$ è la **funzione di transizione**
4. $q_0 \in Q$ è lo **stato iniziale**
5. F $\subseteq Q$ è l'**insieme degli stati accettanti**

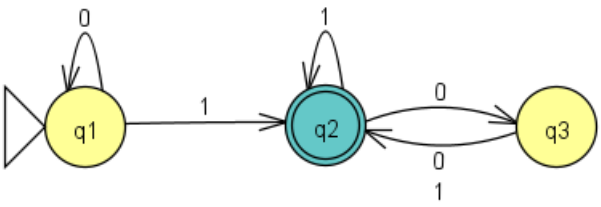


Figure 1.1: L'automa finito M_1

Possiamo descrivere M_1 formalmente ponendo $M_1 = (Q, \Sigma, \delta, q_0, F)$ dove:

1. Q = $\{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. σ è descritta come segue:

	0	1
→q1	q1	q2
∗q2	q3	q2
q3	q2	q2

4. $q_0 = q_1$ è lo stato iniziale
5. F = $\{q_2\}$ sono gli stati accettanti

Se A è l'insieme di tutte le stringhe che la macchina M accetta, diciamo che A è il **linguaggio della macchina M** e scriviamo *L(M) = A*. Inoltre diciamo che ***M riconosce A*** o che ***M accetta A***.
Una macchina può accettare diverse stringhe, ma riconosce sempre e solo un linguaggio. Se la macchina non accetta alcuna stringa, essa riconosce ancora un linguaggio, ossia il linguaggio vuoto $\emptyset$.

Nel nostro esempio, sia:
A = { w | w contiene almeno un 1 e un numero pari di 0 segue l'ultimo 1}

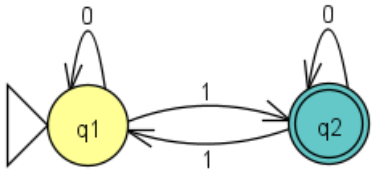
Allora L(M_1) = A, oppure anche, M_1 riconosce A.

1.1.1 Definizione linguaggio regolare

Un linguaggio è detto ***linguaggio regolare*** se esiste un automa finito che lo riconosce.

1.1.2 Esempi DFA

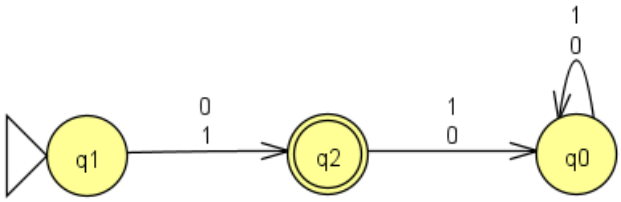
Quale linguaggio riconosce il seguente DFA M_1?



$\mathrm{L}(M_1) = \{w \in \Sigma^* |$ w contiene un numero dispari di 1 $\}$

Esempi di stringhe accettate: 1, 111, 00010000, 001001001000.
Esempi di stringhe rifiutate: 0, 11, 010010, 0010010010100.

Quale linguaggio riconosce il seguente DFA M_2?



$\mathrm{L}(M_2) = \{w \in \Sigma^* \mid w = 1\}$

Esempi di stringhe accettate: 0, 1.
Esempi di stringhe rifiutate: 00, 01, 000, 101.

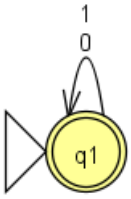
Il complemento di $\mathrm{L}(M_2)$ è il linguaggio di tutte le stringhe su Σ^* che hanno lunghezza diversa da 1.
Cioè $\mathrm{C}(\mathrm{L}(M_2)) = \{w \in \Sigma^* \mid w \neq 1\}$.

Graficamente possiamo porre come nuovi stati finali tutti gli statti che prima non lo erano e poi rendiamo i vecchi statti accettanti come stati normali.
Idea:
$\{w \in \Sigma^* \mid w \neq 1\} = \{\epsilon\} \cup \{w \in \Sigma^* \mid |w| > 1\}$

Nota bene:

Un DFA accetta ϵ se e solo se il suo stato iniziale è accettante

Quale linguaggio riconosce il seguente DFA M_3?



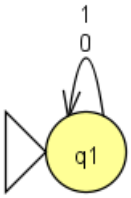
M_3 accetta tutte le possibili stringhe su $\Sigma = \{$a, b$\}$. Cioè:

$$\mathrm{L}(M_3) = \Sigma^*$$

Nota bene:

Un DFA tale che tutti gli stati sono accettanti riconosce il linguaggio Σ^****.***

Quale linguaggio riconosce il seguente DFA M_4?



In questo caso, l'insieme degli stati accettanti è vuoto. M_4 rifiuta tutte le stringhe. Cioè:

$$\mathrm{L}(M_4) = \emptyset$$

Nota bene:

Ogni DFA che non ha stati accettanti rifiuta tutte le stringhe: riconosce il linguaggio $\emptyset$***.***

1.2 Operazioni regolari su DFA

Siano A e B linguaggi. Definiamo le operazioni regolari **unione**, **concatenazione** e **star(Kleene star**) come segue:

- **unione**: $A \cup B = \{x \mid x \in A \text{ o } x \in B\}$
- **concatenazione**: $A \text{ o } B = \{xy \mid x \in A \text{ e } y \in B\}$
- **star**: $A^* = \{x_1x_2x_3...x_k \mid k \geq 0 \text{ e ogni } x_i \in A\}$

1.2.1 Chiusura

Una collezione S di oggetti è chiusa per un'operazione f se applicando f a membri di S, f restituisce un oggetto ancora in S.
Ad esempio N = {0, 1, 2, . . .} è chiuso per addizione, ma non per sottrazione.

Ad esempio abbiamo visto che dato un DFA M_1 che riconosce il linguaggio L, possiamo costruire il DFA M_2 che riconosce il complemento $L' = \text{C(L)}$.

Quindi L regolare $\Rightarrow$ C(L) regolare

1.2.2 Linguaggi regolari chiusi per complemento

L'insieme dei linguaggi regolari è chiuso per l'operazione di complemento.

Dimostrazione:
Sia L un linguaggio regolare su un alfabeto Σ.
Esiste un DFA M = (Q, Σ, δ, q_1, F) che riconosce L.

Il DFA M' = (Q, Σ, δ, q_1, Q F) riconosce $L' = \text{C(L)}$.

Perchè funziona?
Ogni stringa in L è accettata da M (M termina in uno stato in q $\in$ F) e quindi rifiutata da M' (perché q $\notin Q -$ F).

Ogni stringa in C(L) è rifiutata da M e quindi è accettata da M_0.

$$M_0 \text{ accetta x} \Longleftrightarrow \in \text{C(L)}$$

1.2.3 Linguaggi regolari chiusi per 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

Dimostrazione:
L_1 ha un DFA M_1. L_2 ha un DFA M_2.
Una stringa w è in $L_1 \cup L_2$ se e solo se w è accettata da M_1 oppure da M_2. Bisogna definire un DFA M_3 che accetti w se e solo se w è accettata da M_1 o M_2.

M_3 dovrà essere capace di:

- tener traccia di dove l'input sarebbe se fosse contemporaneamente in input a M_1 e M_2
- accettare una stringa w se e solo se M_1 oppure M_2 accettano la stringa

Siano L_1 e L_2 due linguaggi definiti sull alfabeto Σ.

Sia $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ il DFA che riconosce L_1
e sia $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ il DFA che riconosce L_2

Costruiamo il DFA $M_3 = (Q_3, \Sigma_3, \delta_3, q_3, F_3)$ formato da:

- $Q_3 = Q_1 \times Q_2 = \{(x, y) \mid x \in Q_1 \text{ e } y \in Q_2\}$
- Σ rimane lo stesso
- $\delta_3 = Q_3 \times \Sigma \longrightarrow Q_3$ per ogni $x \in Q_1$, $y \in Q_2$ e ogni $a \in \Sigma$
  $\delta_3((x, y), a) : (\delta_1(x, a), \delta_2(y, a))$
- $q_3 =$ è la coppia $\{q_1, q_2\}$
- $F_3 = \{(x, y) \mid x \in F_1 \text{ o } y \in F_2\}$ basta che uno dei due è accettante

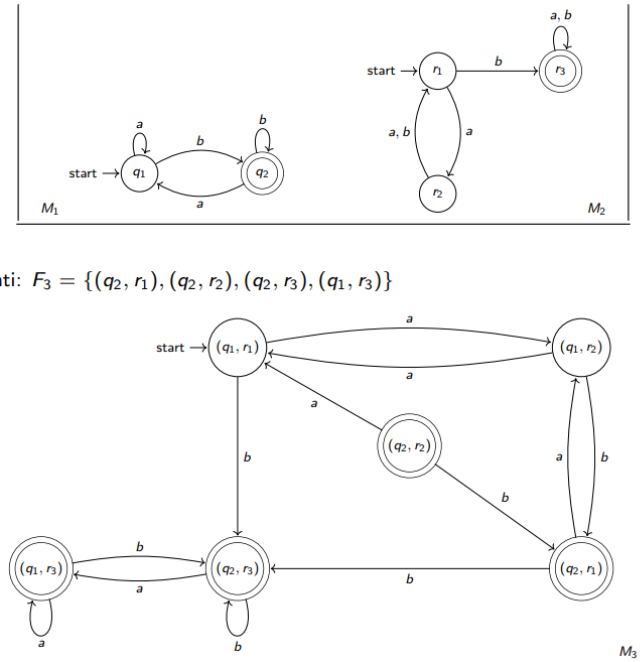


Figure 1.2: Unione di 2 DFA

1.2.4 Linguaggi regolari chiusi per intersezione

Analogamente all'unione, possiamo ragionare allo stesso modo per l'intersezione di due linguaggi regolari.

La classe dei linguaggi regolari è chiusa per l'operazione di intersezione. Cioè, se L_1 e L_2 sono linguaggi regolari, allora lo è anche $L_1 \cap L_2$.

Idea:
Per far si che una macchina M accetti una stringa w su 2 linguaggi L_1 e L_2 dobbiamo far in modo che venga accettata sia la macchina M_1 e sia la macchina M_2.
In poche parole rispetto all'unione cambia solo l'insieme degli stati accettati dove ora abbiamo che F = $\{(x, y) \mid x \in F_1 \text{ e } y \in F_2\}$, quindi tutti e due gli stati della coppia devono essere stati accettanti nel loro automa di partenza.

1.2.5 Linguaggi chiusi per concatenazione

La classe dei linguaggi regolari è chiusa per l'operazione di concatenazione. Cioè, se L_1 e L_2 sono linguaggi regolari, allora lo è anche L_1 o L_2.
Come dimostrarlo?
Si potrebbe procedere come fatto finora: partire da un DFA M_1 che riconosce L_1 e un DFA M_2 che riconosce L_2 e costruire un DFA M_3 che riconosca L_1 o L_2.

Come costruire M_3?
L'automa M_3 dovrebbe accettare una stringa w se e solo se essa può essere divisa in due parti tali che la prima parte è accettata da M_1 e la seconda parte è accettata da M_2.

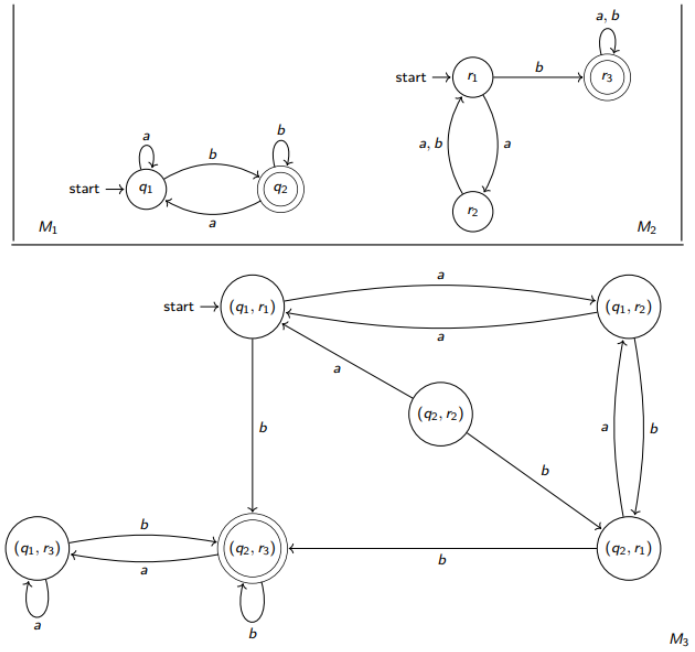


Figure 1.3: Intersezione di 2 DFA

Il problema è che M_3 non sa dove finisce la prima parte e dove comincia la seconda. Si dovrebbero analizzare tutte le possibilità:
troppo laborioso! Lasciamo per il momento in sospeso il problema e introduciamo un nuovo argomento:
il non determinismo

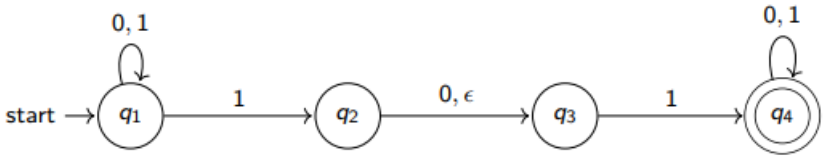
1.3 Automi finiti non deterministici

La computazione deterministica ci dice che per ogni simbolo esiste una sola transizione a partire da un certo stato.
Quindi lo stato successivo è **univocamente determinato**.

Formalmente:
Un **DFA** $M = (Q, \Sigma, \delta, q_0, F)$: per ogni stato $q \in Q$ e per ogni simbolo $a \in \Sigma$ letto, esiste un **unico stato(Determinismo)** in cui si transisce $\delta : Q \times \Sigma \longrightarrow Q$.

Computazione non deterministica
Il non determinismo è una generalizzazione del determinismo.Infatti un automa finito non deterministico(**NFA**) permette di avere:

- 0, 1 o anche più archi uscenti per ciascun simbolo dell'alfabeto
- 0, 1 o anche più archi uscenti con etichetta ϵ (epsilon-transition)



[?]

Come computa un NFA?

Se un NFA è in uno stato e, leggendo un simbolo a dell'alfabeto , si trova davanti a più scelte (più archi uscenti con lo stesso simbolo a):

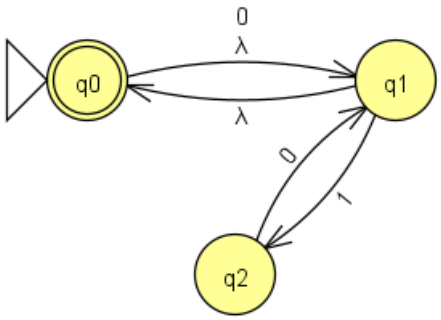
- si divide in più copie di se stesso: una copia per ciascuna scelta
- ogni copia segue la computazione in parallelo indipendentemente dalle altre

Se una copia di un NFA legge un simbolo per il quale non vi è un arco uscente dallo stato in cui si trova in quel momento, allora si dice che la copia della macchina **muore** insieme a tutto il suo ramo.

Una stringa w può sessere accettata da un NFA:

- se almeno una copia raggiunge uno stato accettante
- invece se nessuna copia raggiunge uno stato accettante la stringa w viene rifiutata

Nota bene:
Se abbiamo un ciclo formato da ϵ-archi non vengono generate infinite copie di una macchina perché ogni volta il numero di copie che si creano sono pari al numero di stati in cui la macchina transisce, quindi un numero finito di stati.



In questo caso all'inizio della computazione non vengono generate infinite copie ma solo 2.Non ha senso far andare più di 2 macchine sullo stesso input.

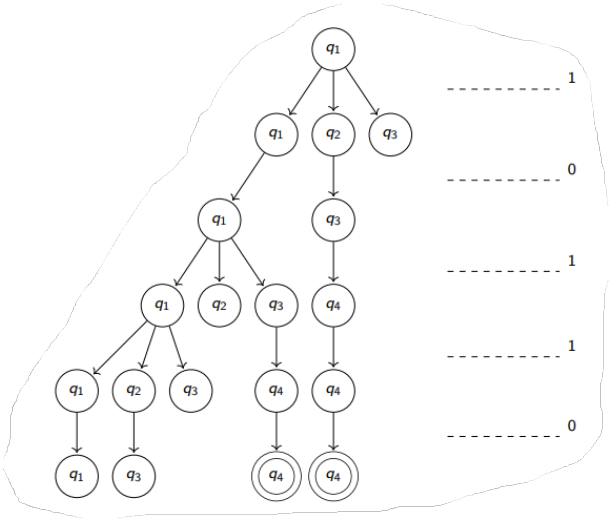
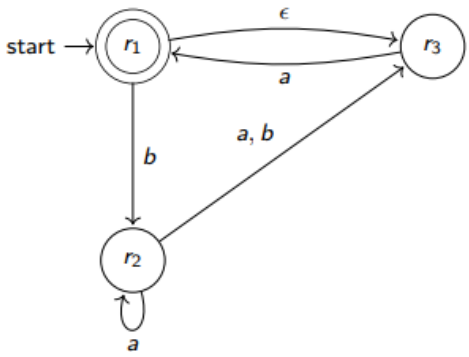


Figure 1.4: Albero di computazione di un NFA

1.3.1 Esempi NFA

Dato il seguente NFA N_1 determinare se alcune stringhe appartengono al linguaggio L di N_1.



Stringa input: a
a accettata e quindi appartenete a L(N_1) perchè:

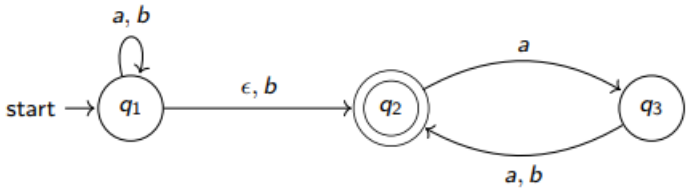
- Si parte da r_1 come stato iniziale e la computazione si duplica subito dato che vi è la presentza di un ϵ-arco che porta a r_3.
- La copia rimasta in r_1 muore dato che non esiste una transizione con il simbolo a.
- La copia che parte da r_3 invece, ha la transizione del simbolo a che porta a r_1.
- Dato che lo stato r_1 è uno stato accettante e l'input è stato consumato tutto possiamo dire che la stringa a è accettata.

Stringa input: ϵ
ϵ accettata perchè r_1 è accettante.

Alcune stringhe accettate: aa, baa, baba.
Alcune stringhe rifiutate: b, ba, bb.

1.4 Definizione NFA

Un automa finito **non deterministico** è una quintupla $N = (Q, \Sigma, \delta, q_0, F)$ dove:



- Q è un insieme finito degli **stati**
- Σ è un insieme finito chiamato **alfabeto**
- $\delta : Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$ è la **funzione di transizione** Sia $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
- $q_0 \in Q$ è lo **stato iniziale**
- F $\subseteq Q$ è l'**insieme degli stati accettanti**

La differenza tra DFA e NFA sta nella definizione della funzione di transizione. In un NFA la funzione δ

- ammettere mosse di tipo ϵ(ϵ-transizioni)
- restituisce un insieme di stati (notare che $\emptyset \in \mathcal{P}(Q)$: nell'esempio $\delta(q_2, b) = \emptyset$)

Osservazione:
Il nondeterminismo è una generalizzazione del determinismo: ogni DFA è anche un NFA

1.4.1 Computazione di un NFA

Sia $N = (Q, \Sigma, \delta, q_0, F)$ un NFA.
Consideriamo la stringa $w = w_1 w_2 ... w_n$ con $w_i \in \Sigma$ per $i = 1, 2, ..., n$.
N accetta la strigna w se w può essere scritto come $w = y_1 y_2 ... y_m$ dove ciascun $y_i \in \Sigma_\epsilon$, e esiste una sequenza di stati $r_0 r_1 ... r_m$ tale che:

- $r_0 = q_0$
- $r_{i+1} \in \delta(r_i, y_{i+1})$ per ogni $i = 1, 2, ..., m-1$
- $r_m \in F$

Se A è l'insieme di di tutte le stringhe che la macchina accetta, diciamo che A è il linguaggio della macchina N e scriviamo $L(N) = A$.

Diciamo che N **riconosce** A.

1.4.2 Equivalenza tra DFA e NFA

*Due macchine sono **equivalenti** se riconoscono lo stesso linguaggio.*

Ogni **NFA** N ha un equivalente **DFA** M; cioè, se N è un NFA, **allora esiste un DFA** M **tale che** $L(M) = L(N)$.

Come dimostrarlo?
Bisogna far vedere che a partire da un NFA N sia in grado di costruire un DFA M che sia in grado di emulare N.

NFA a DFA senza ϵ-archi

Partiamo da un generico NFA $N = (Q_N\Sigma, \delta_N, q_N, F_N)$.
Costruiamo un DFA $M = (Q_M, \Sigma, \delta_M, q_M, F_M)$ tale che $L(M) = L(N)$.

- $Q_M = \mathcal{P}(Q)$ ogni stato di M corrisponde a un insieme di stati di N.
- Quando M si trova in uno stato $R \in Q_M$ e legge il simbolo $a \in \Sigma$, transisce nello stato $R' = \{q \in Q_N \mid q \in \delta_N(r, a), r \in R\}$ cioè:
$$\delta_M(R, a) = \bigcup_{r \in R} \delta_N(r, a)$$
- $q_M = \{q_N\}$
- $F_M = \{R \in Q_M \mid R \cap F_N \neq \emptyset\}$

NFA a DFA con ϵ-archi

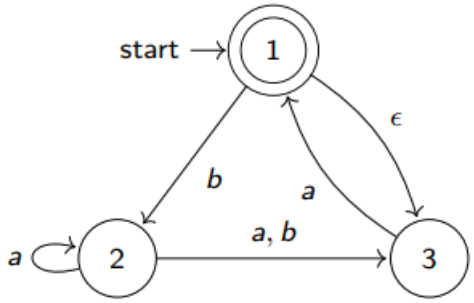
Ora vediamo il caso generale, ovvero quando un NFA contiene uno o più ϵ-archi.

Partiamo da un generico NFA $N = (Q_N\Sigma, \delta_N, q_N, F_N)$.
Costruiamo un DFA $M = (Q_M, \Sigma, \delta_M, q_M, F_M)$ tale che $L(M) = L(N)$.

Dato uno stato $R \in Q_M$, definiamo l'insieme di tutti gli stati che possono essere raggiunti dagli elementi di R usando 0 o più ϵ-archi.
$E(R) = \{q \in Q_n \mid q$ può essere raggiunto da R usando 0 o più ϵ-archi $\}$. **Epsilon chiusura**

- $Q_M = \mathcal{P}(Q)$ ogni stato di M corrisponde a un insieme di stati di N. **(come prima)**
- Quando M si trova in uno stato $R \in Q_M$ e legge il simbolo $a \in \Sigma$, transisce nello stato $R' = \{q \in Q_N \mid q \in E(\delta_N(r, a)), r \in R\}$ cioè:
$$\delta_M(R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$$
- $q_M = E(\{q_n\})$
- $F_M = \{R \in Q_M \mid R \cap F_N \neq \emptyset\}$ **(come prima)**

Esempio Da NFA a DFA equivalente



Vogliamo trasformare il seguente NFA in un DFA equivalente.

Innanzitutto definiamo questo NFA come $N = (Q_N\Sigma, \delta_N, q_N, F_N)$ dove:

- $Q_N = \{1, 2, 3\}$
- $\Sigma = \{a, b\}$
- $\delta_N = Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$:

	a	b	ϵ
$\rightarrow *1$	$\emptyset$	$\{2\}$	$\{3\}$
2	$\{2,3\}$	$\{3\}$	$\emptyset$
3	$\{1\}$	$\emptyset$	$\emptyset$

- $q_N = 1$
- $F_N = \{1\}$

Noi vogliamo avere un DFA $M = (Q, \Sigma, \delta, q_0, F)$ equivalente a N con:

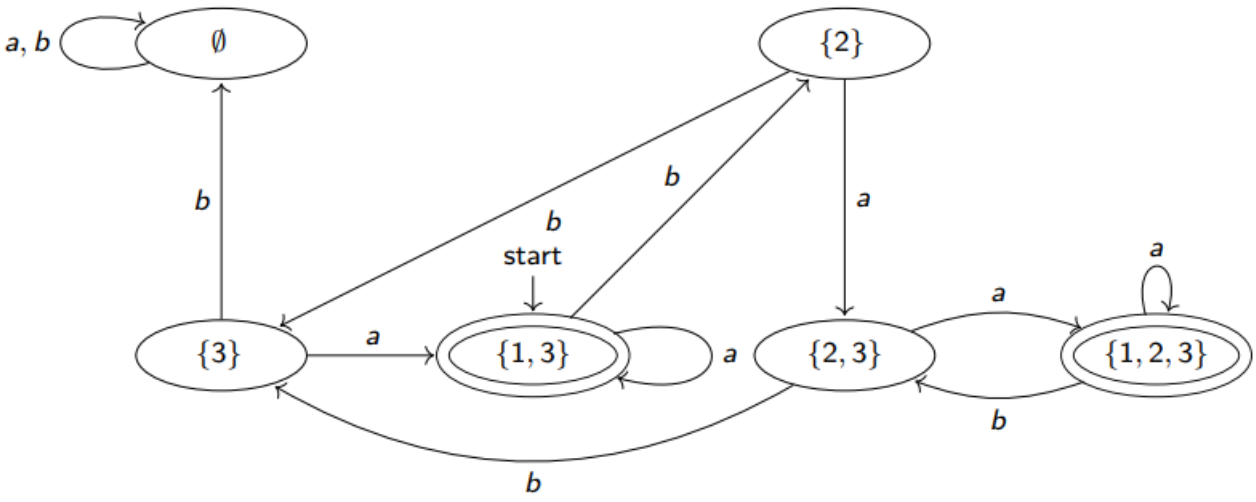
- $Q = \mathcal{P}(Q)$
- $\delta = \{q \in Q_N \mid q \in E(\delta_N(r, a)), r \in R\}$ con R si indica uno stato $\in Q$
- $q_0 = E(\{1\})$
- $F_N = \{R \in Q \mid R \cap F_N \neq \emptyset\}$

Data la descrizione formale, ora ci tocca procedere per passi per poter disegnare o scrivere la funzione di transizione del nostro DFA equivalente al nostro NFA di partenza.

- Per prima cosa bisogna capire qual'è il nuovo stato iniziale. Per fare ciò ci basta fare l'epsilon chiusura dello stato iniziale del nostro NFA.
$$E(\{1\} = \{1, 3\}$$
- Successivamente si procede facendo δ delle stato iniziale appena individuato per tutti i simboli appartenenti all'alfabeto.(nel nostro caso $\Sigma = \{a, b\}$)

$$\delta((\{1, 3\}), a) = E(\delta_N(1, a)) \cup E(\delta_N(3, a)) = \emptyset \cup E\{1\} = \{1, 3\}$$
$$\delta((\{1, 3\}), b) = E(\delta_N(1, b)) \cup E(\delta_N(3, b)) = E\{2\} \cup \emptyset = \{2\}$$

- Ora andiamo avanti fino a quando scopriamo nuovi stati nello stesso modo(ad esempio in questo passo abbiamo scoperto lo stato $\{2\}$, di cui faremo la transizione per a e per b.
- Alla fine quando abbiamo scoperto tutti gli stati dobbiamo marcare come stati accettanti tutti gli stati che contengono uno stato accettante del nostro NFA di partenza.



1.4.3 Linguaggi regolari e NFA

*Un linguaggio è **regolare** se e solo se esiste un **automa finito non deterministico** che lo riconosce.*

Dimostrazione:
Sia L un linguaggio regolare. Allora (per definizione) esiste un DFA che lo riconosce. Ma ogni DFA è anche un NFA, quindi esiste un NFA che riconosce L. Quindi:

$$L \text{ è regolare} \implies \text{esiste un NFA che riconosce } L.$$

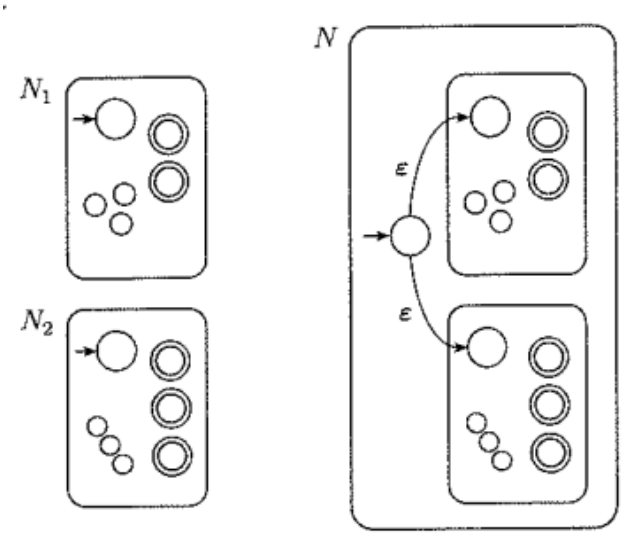
Sia N un NFA che riconosce un linguaggio L. Abbiamo dimostrato che ogni NFA ha un equivalente DFA. Quindi esiste un DFA M che riconosce lo stesso linguaggio L. Quindi, per definizione, L è regolare. Quindi:

$$\text{esiste un NFA che riconosce } L \implies \text{è regolare}$$

1.4.4 Linguaggi regolari chiusi per unione

La classe dei linguaggi regolari è chiusa rispetto all'operazione di unione.

Idea:
Abbiamo i linguaggi regolari A_1 e A_2 e vogliamo provare che $A_1 \cup A_2$ è regolare. L'idea è di prendere 2 NFA, N_1 e N_2, e i rispettivi linguaggi,A_1 e A_2 e comporli in un un nuovo NFA, N. La macchina N deve accettare il suo input se $N1$ o $N2$ accetta l'input.
La nuova macchina ha uno stato iniziale che si dirama nei 2 stati iniziali delle vecchie macchine con ϵ-archi. In questo modo la macchina accetta se una delle 2 macchine accetta l'input.



Possiamo dimostrare ciò tramite la definizione formale:

Siano L_1 e L_2 due linguaggi definiti sull alfabeto Σ.

Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ il NFA che riconosce A_1
e sia $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ il NFA che riconosce A_2

Costruiamo l' NFA $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere $A_1 \cup A_2$:

- $Q = \{q_0\} \cup Q_1 \cup Q_2$
  Gli stati di N sono tutti gli stati di N_1 e N_2 con l'aggiunta di uno stato iniziale q_0
- Σ rimane lo stesso
- Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\epsilon$:

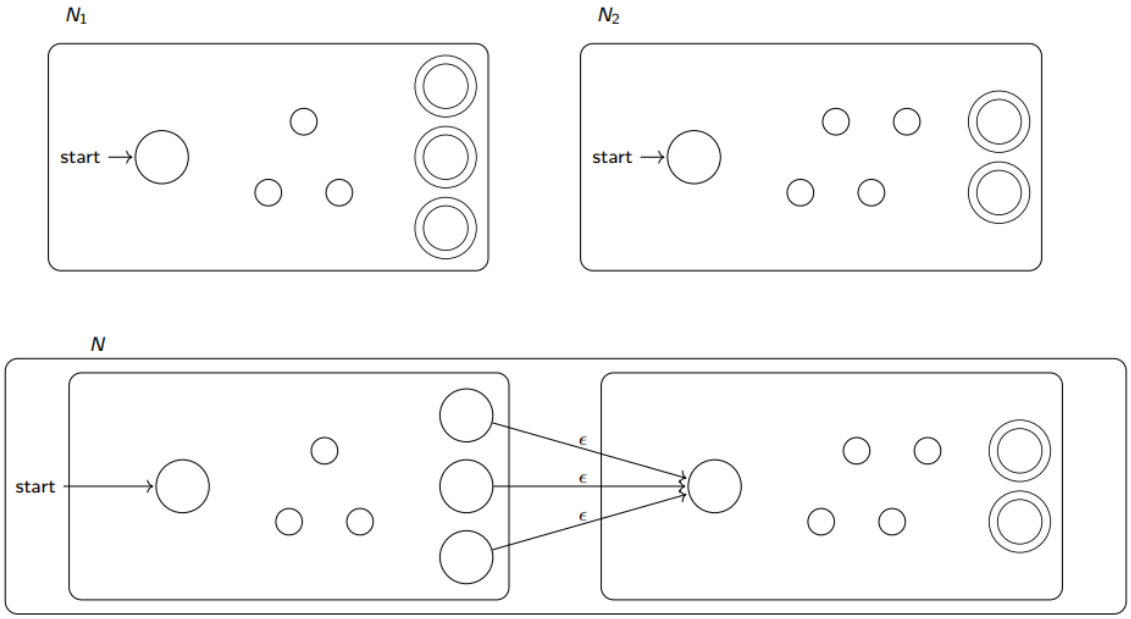
$$\delta = \left\{ \begin{array}{lll} \delta_1(q,a) & \text{se} & q \in Q_1 \\ \delta_2(q,a) & \text{se} & q \in Q_2 \\ \{q_1, q_2\} & \text{se} & q = q_0 \text{ e } a = \epsilon \\ \emptyset & \text{se} & q = q_0 \text{ e } a \neq \epsilon \end{array} \right\}$$

- q_0 è lo stato iniziale di N
- $F = F_1 \cup F_2$
  Gli stati accettanti di N sono tutti gli stati accettanti di N_1 e N_2.

1.4.5 Linguaggi regolari chiusi per concatenazione

La classe dei linguaggi regolari è chiusa rispetto all'operazione di concatenazione.

Idea:
Abbiamo i linguaggi regolari A_1 e A_2 e vogliamo provare che $A_1 \circ A_2$ è regolare. L'idea è di prendere 2 NFA, N_1 e N_2, e i rispettivi linguaggi,A_1 e A_2 e comporli in un un nuovo NFA, N. La macchina N deve avere come stato iniziale quello della macchina N_1. Gli stati accettanti di N_1 avranno degli ϵ-archi che puntano allo stato iniziale di N_2, quindi ogni volta che ci troviamo in uno stato accettante di N_1 ci dirameremo direttamente in N_2.
Gli stati accettanti di N sono gli stati accettanti di N_2.



Possiamo dimostrare ciò tramite la definizione formale:

Siano L_1 e L_2 due linguaggi definiti sull'alfabeto Σ.

Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ il NFA che riconosce A_1
e sia $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ il NFA che riconosce A_2

Costruiamo l' NFA $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere $A_1 \circ A_2$:

- $Q = Q_1 \cup Q_2$
  Gli stati di N sono tutti gli stati di N_1 e N_2.
- Σ rimane lo stesso
- Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\epsilon$:

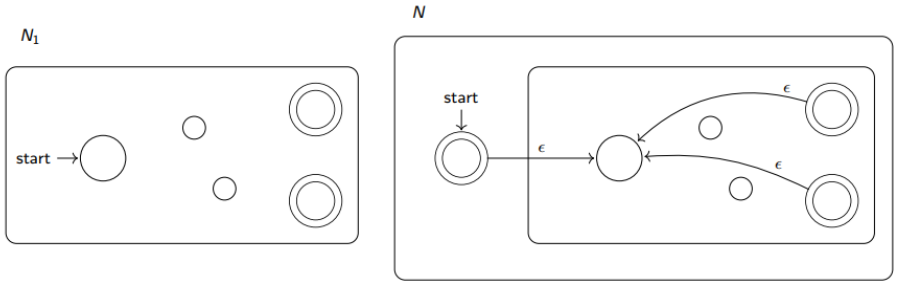
$$\delta = \left\{ \begin{array}{ll} \delta_1(q,a) & \text{se } q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \text{ e } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_2\} & \text{se } q \in F_1 \text{ e } a = \epsilon \\ \delta_2(q,a) & \text{se } q \in Q_2 \end{array} \right\}$$

- q_1 è lo stato iniziale di N ed è lo stesso di N_1
- $F = F_2$ sono gli stati accettanti di N_2

1.4.6 Linguaggi regolari chiusi per star

La classe dei linguaggi regolari è chiusa rispetto all'operazione di star.

Idea:
Abbiamo il linguaggio regolare A_1 e vogliamo provare che A_1^* è regolare. Prendiamo un NFA N_1 per A_1 e lo modifichiamo per riconoscere A_1^*.
L'NFA che otteniamo N, accetterà il suo input quando può essere diviso in varie parti ed N_1 accetta ogni parte. Possiamo costruire N come N_1 con vari ϵ-archi che vanno dagli stati finali di N_1 allo stato iniziale. Inoltre dobbiamo modificare N in modo che accetti ϵ, ci basta aggiungere un nuovo stato iniziale, che è uno stato accettante, e ha un ϵ-arco verso il vecchio stato iniziale di N_1.



Possiamo dimostrare ciò tramite la definizione formale:

Siano L_1 un linguaggio definito sull'alfabeto Σ.
Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ l'NFA che riconosce A_1

Costruiamo l'NFA $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere A_1^*:

- $Q = \{q_0\} \cup Q_1$
  Gli stati di N sono quelli si N_1 più il nuovo stato iniziale
- Σ rimane lo stesso
- Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\epsilon$:

$$\delta = \left\{ \begin{array}{ll} \delta_1(q,a) & \text{se } q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \text{ e } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{se } q \in F_1 \text{ e } a = \epsilon \\ \{q_1\} & \text{se } q = q_0 \text{ e } a = \epsilon \\ \emptyset & \text{se } q = q_0 \text{ e } a \neq \epsilon \end{array} \right\}$$

- q_0 è lo stato iniziale di N
- $F = F_1 \cup \{q_0\}$
  Gli stati accettanti di sono quelli di N_1 più il nuovo stato accettante

1.5 Espressioni Regolari

Abbiamo visto che un **automa a stati finiti**(DFA o NFA) è un modo per definire i **linguaggi regolari**.
Vedremo che una **espressione regolare** è un modo dichiarativo per descrivere un linguaggio regolare.
Consideriamo l'alfabeto $\Sigma = \{0, 1\}$.
Nelle espressioni regolari per brevità:

- 0 indica l'insieme $\{0\}$;
- 1 indica l'insieme $\{1\}$.

Quindi l'espressione $0 \cup 1$ indica $\{0\} \cup \{1\}$, cioè $\{0, 1\}$. L'espressione $(0 \cup 1)0^*$ corrisponde a $\{0, 1\} \circ \{0\}^*$. Questo è l'insieme delle stringhe binarie che iniziano con 0 oppure 1 e continuano con degli 0 (anche nessuno).
L'espressione $((0 \cup 1)^*$ corrisponde a $\{0, 1\}^*$, cioè l'insieme di tutte le stringhe binarie.
Vediamo ora di dare una definizione formale di espressione regolare.

Definizione Induttiva:
Base:

- a è un'espressione regolare per ogni a $\in \Sigma$
- ϵ è un'espressione regolare (denota $\{\epsilon\}$)
- $\emptyset$ è un'espressione regolare.

Passo:

- se R_1 e R_2 sono espressioni regolari, allora $R_1 \cup R_2$ è un'espressione regolare;
- se R_1 e R_2 sono espressioni regolari, allora $R_1 \circ R_2$ è un'espressione regolare;
- se R è un'espressione regolare, allora R^* è un'espressione regolare.

La definizione induttiva suggerisce un metodo per dimostrare che una espressione regolare R soddisfa una certa proprietà.

Base Si dimostra che la proprietà è soddisfatta per i casi base:

è soddisfatta per $R = a$, dove $a \in \Sigma$;

è soddisfatta per $R = \epsilon$

è soddisfatta per $R = \emptyset$

Passo Si suppone per ipotesi induttiva che la proprietà sia soddisfatta da R_1 e R_2 e si dimostra che è soddisfatta anche da ciascuna delle seguenti espressioni regolari:

$R = R_1 \cup R_2$;

$R = R_1 \circ R_2$;

$R = R^*$

Le operazioni regolari hanno il seguente **ordine di precedenza**:

1. Kleene star
2. Concatenazione
3. Unione

Le parentesi cambiano l'ordine.

1.5.1 Teorema di Kleene

Espressioni regolari e automi finiti sono equivalenti nella loro potenza descrittiva.

Ogni espressione regolare può essere convertita in un automa finito che riconosce il linguaggio che essa descrive e viceversa.
Data una espressione regolare R indichiamo con $L(R)$ il linguaggio descritto da R.

Un linguaggio è regolare se e solo se esiste una espressione regolare che lo descrive.

Idea: Sappiamo che un linguaggio L è regolare se e solo se L è riconosciuto da un NFA se e solo se L è riconosciuto da un DFA

Dimostriamo che:

- per ogni espressione regolare R esiste un NFA N, tale che $L(N) = L(R)$;
- per ogni DFA M possiamo costruire un'espressione regolare R con $L(R) = L(M)$.

Cioè L è riconosciuto da un DFA **se e solo se** L può essere descritto da un'espressione regolare.

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

Dimostrazione: Supponiamo di avere un'espressione regolare R che descrive un linguaggio A. Trasformiamo R in un NFA che riconosce A. Questo implica che A è regolare.

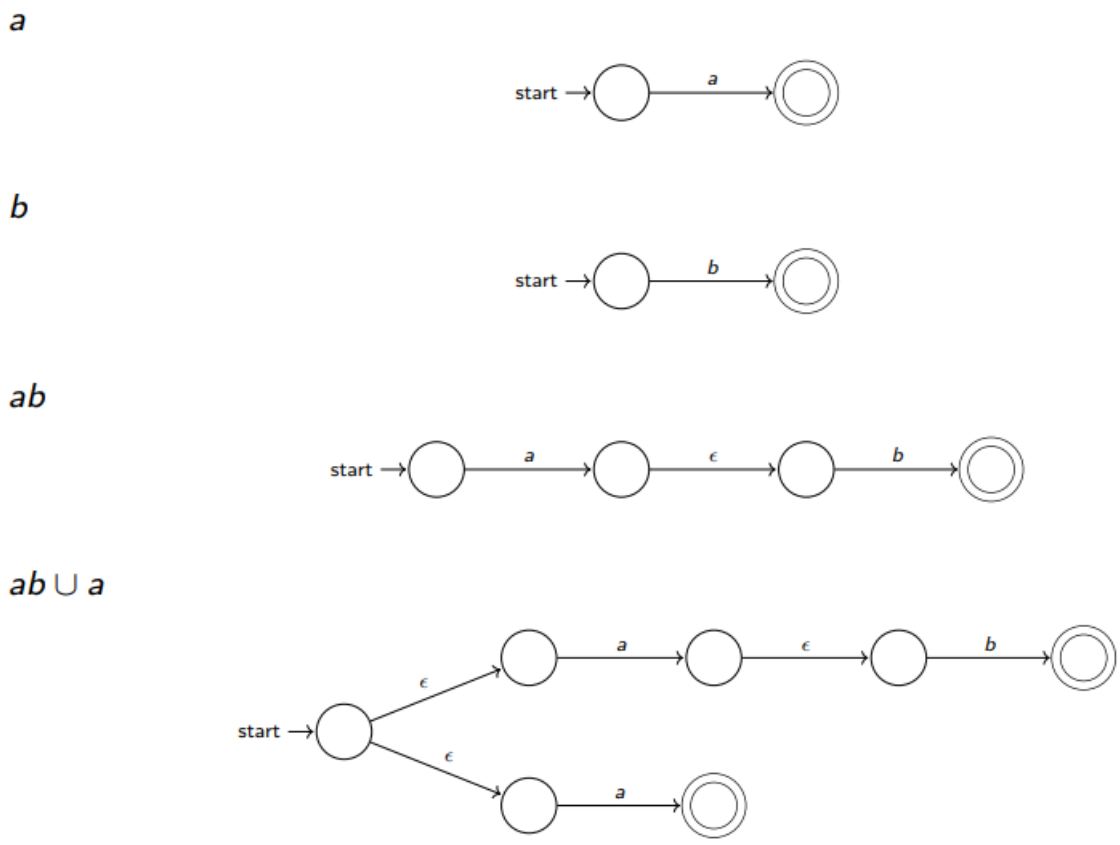
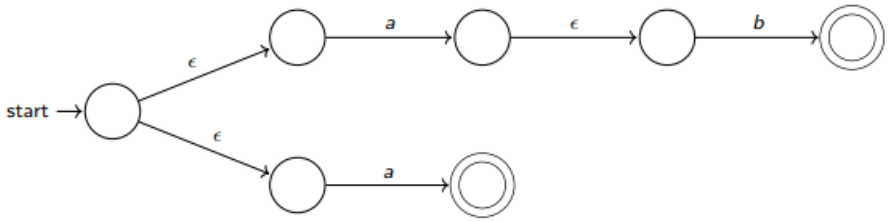


Figure 1.5: Leggenda REGEX

$ab \cup a$



$(ab \cup a)^*$

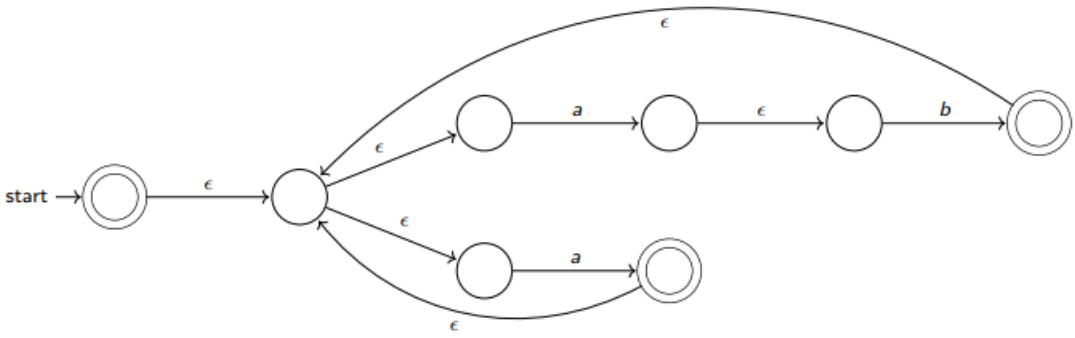
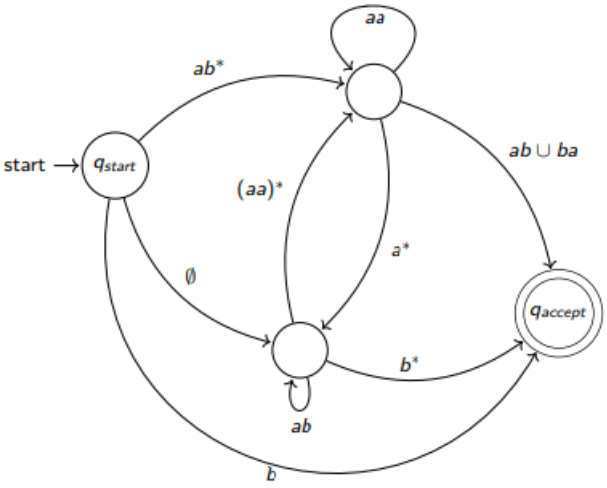


Figure 1.6: Star con REGEX

1.5.2 Da automa a espressione regolare

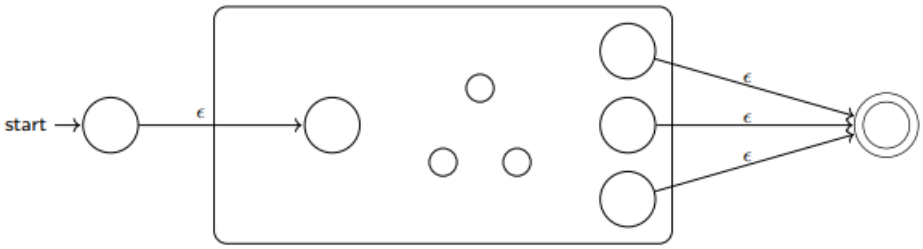
Se un linguaggio è regolare, allora è descritto da un'espressione regolare.

Dimostrazione:
Descriviamo una procedura per trasformare i DFA in espressioni regolari equivalenti.
Useremo una generalizzazione dell'NFA: automa finito non deterministico generalizzato (GNFA) che è un NFA che permette archi etichettati con espressioni regolari.



Convertiamo un DFA in un GNFA in forma speciale.

1. Aggiungiamo un nuovo stato iniziale con un ϵ-arco verso il vecchio stato iniziale
2. Aggiungiamo un nuovo stato accettante con ϵ-archi entranti provenienti dai vecchi stati accettanti.



Eventuali archi con più etichette o archi che collegano 2 stati nella stessa direzione vengono sostituiti con un solo arco etichettato con l'unione delle precedenti etichette.

Abbiamo così ottenuto un GNFA in forma speciale equivalente al DFA di partenza.

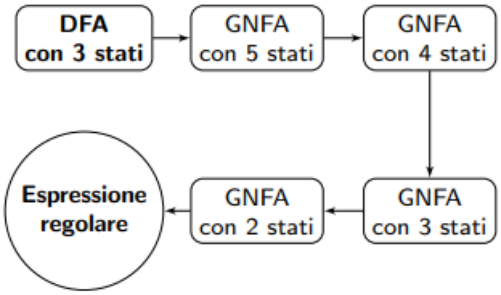
Trasformiamo ora questo GNFA in un'espressione regolare equivalente.

Sia k il numero di stati del GNFA. Chiaramente $k \geq 2$, visto che stato iniziale e stato accettante sono diversi.
Il nostro obiettivo è di convertire questo GNFA con $k \geq 2$ in un GNFA equivalente con 2 stati.

Per $k = 2$ il GNFA ha un solo arco dallo stato iniziale allo stato finale. L'etichetta di questo arco è l'espressione regolare equivalente (e la conversione è terminata).

1.5.3 Metodo per ottenere l'espressione regolare da un automa

Conversione da $k \geq 2$ stati a $k = 2$ stati. Se $k > 2$ costruiamo un GNFA equivalente con $k - 1$ stati.
Iteriamo la procedura fino ad arrivare ad un GNFA con $k = 2$ stati.
I passi della procedura di conversione partendo da un DFA con 3 stati.



Scegliamo uno stato qualsiasi che sia diverso da quello iniziale e da quello accettante e lo togliamo dalla macchina modificando tutto il resto in modo che la nuova macchina riconosca lo stesso linguaggio.

Sia q_{rip} lo stato rimosso. La macchina viene modificata nel modo seguente. Si cambiano le espressioni regolari che etichettano i restanti archi in modo da controbilanciare l'assenza di q_{rip} reintegrando le computazioni rimosse.

Ogni nuova etichetta che va da uno stato q_i a uno stato q_j è un'espressione regolare che descrive tutte le stringhe che avrebbero portato la macchina da q_i a q_j direttamente o attraverso q_{rip}.

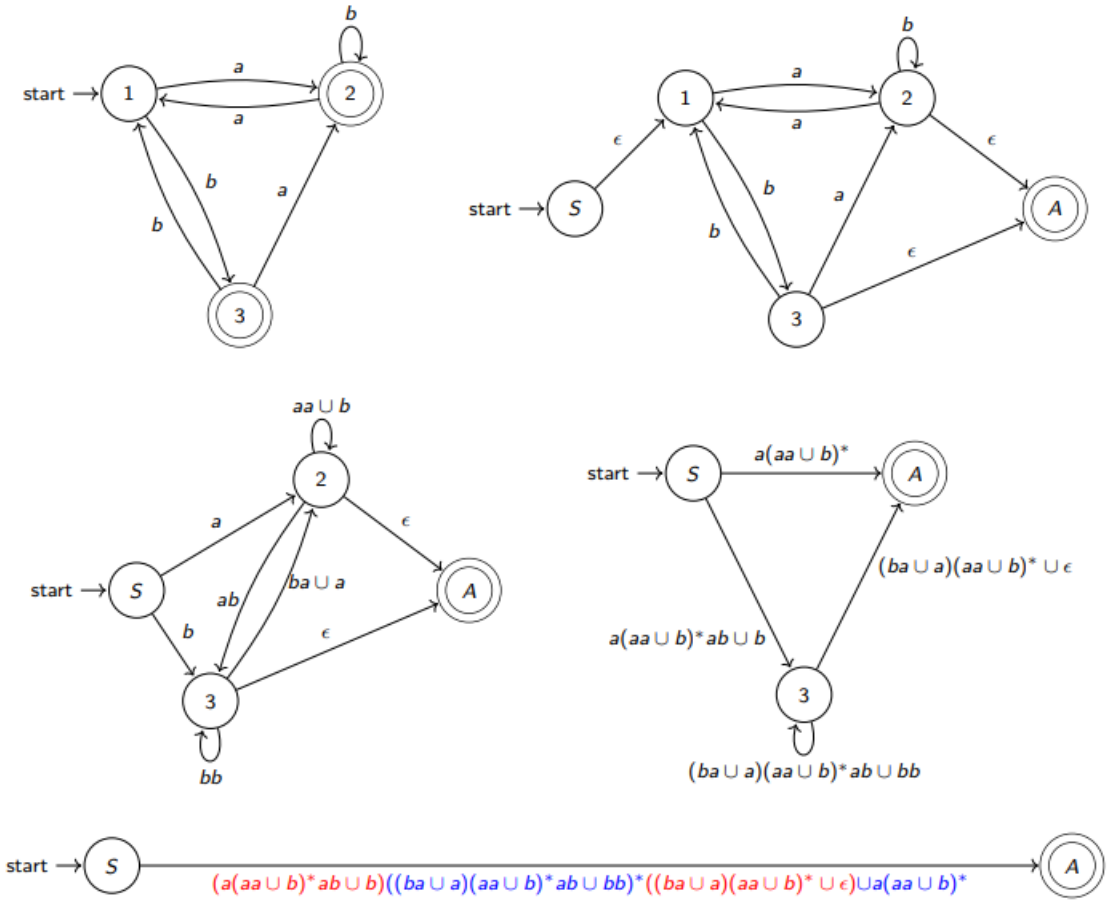


Figure 1.7: Esempio GNFA to regular expression

1.6 Pumping Lemma

Chapter 2

Macchine di Turing

2.1 Introduzione alle MdT

DFA/NFA pregi: estremamente semplici ed economici da costruire. Particolarmente adatti ad alcuni compiti come - patter,matching,lavastoviglie telecomandi....

DFA/NFA difetti: non sono sufficientemente "potenti" per risolvere problemi importanti

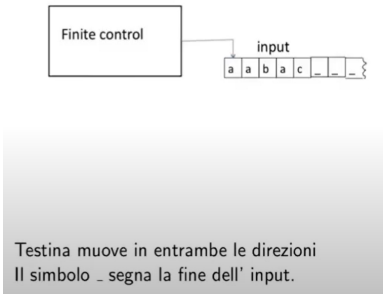


Figure 2.1: Macchina di Turing

2.2 Macchina di Turing

Una macchina di Turing è una settupla (Q,$\Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}$)

- I**nsieme stati Q**
- **Alfabeto di lavoro** $\Sigma(\_ \notin \Sigma)$
- Γ: **Alfabeto del nastro** $(\_ \in \Gamma, \Sigma \subset \Gamma)$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ **funzione di transizione**
- q_0: **stato iniziale**
- q_{accept}: **stato di accettazione**
- q_{reject}: **stato di rifiuto**

Una differenza dagli automi e che quando raggiunge uno stato di accept o reject un MdT si ferma subito.

- Ad ogni instante M occupa uno degli stati in Q
- La testina si trova in un quadrato del nastro contenente un qualche simbolo $\gamma \in \Gamma$
- La funzione di transizione $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ dipende dallo stato q e dal simbolo di nastro γ

Il range della funzione di transizione sono triple (q', γ', d) con:

- $q' \in Q$
- $\gamma' \in \Gamma$ è il simbolo scritto dalla testina sulla cella del nastro su cui la testina si trova **ALL'INIZIO** della transizione
- $d \in \{L, R\}$ è la direzione in cui la testina si muove

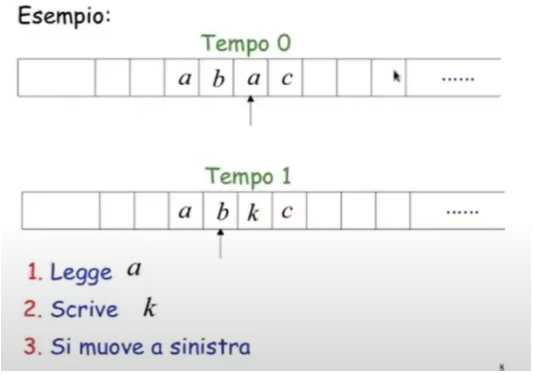


Figure 2.2: Esempio MdT

La computazione parte sempre da uno stato iniziale q_0 e con un input posizionato sulla parte più a sinistra del nastro: le prime n celle, se n è la lunghezza input.

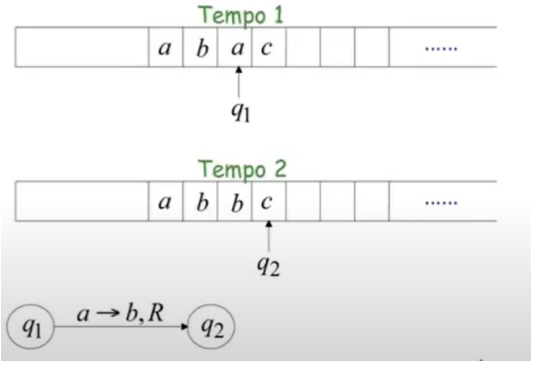


Figure 2.3: Esempio funzione di transizione

La computazione termina quando M raggiunge:

- stato accettazione q_{accept}: **Computazione Accept**
- stato rifiuto q_{reject}: **Computazione Reject**

Una macchina di Turing deterministica è una macchina che quando legge un dato simbolo e si trova in uno stato transisce in un solo stato e non si sono ϵ transizioni

2.2.1 Strategia per accettare $\{a^n b^n\}$

L'idea è quella di cancellare ripetutamente la prima occorrenza di a e l'ultima di b, se la stringa era del tipo $a^n b^n$ non rimango simboli.

Si può fare in 5 passi:

1. Se leggi _, vai a 5. Se leggi a, scrivi _ e vai a 2.
2. Spostati a destra(R) ti tutti a e b. Al primo _, muovi a sinistra(L) e vai a 3.
3. se leggi b, scrivi _ e vai a 4.
4. Spostati a sinistra(L) di tutti a e b. Leggendo _, muovi a destra(R) e vai a 1.
5. Accept.

2.2.2 Configurazioni di MdT

Una Configurazione di una MdT è come una fotografia in un determinato passo. Con fotografia intendiamo che la MdT fermata in un certo istante avrà la testina posizionata su un certo simbolo del nastro, e ci sarà una certa sottostringa a destra della testina e una certa sottostrigna a sinistra della testina.

$$C = uqv$$

- Stato di M è q
- u e v sono le sottostringhe
- la testina è posizionata sul primo simbolo di v e si trova nello stato q (su primo blank _ se v = ϵ)
- contenuto nastro è uv
- dopo v abbiamo solo blanks _

Diciamo che una Configurazione C_1 produce C_2 ($C_1 \longrightarrow C_2$) per via della funzione di transizione.

Esempio: Se a,b,c $\in \Gamma$, u,v $\in \Gamma^*$, $q_i, q_j \in Q$

u$a q_i$bv $\longrightarrow$ u$q_j$acv se $\delta(q_i, b) = (q_j, c, L)$ **Mossa a sinistra**
u$a q_i$bv $\longrightarrow$ uacq_jv se $\delta(q_i, b) = (q_j, c, R)$ **Mossa a destra**

Se la testina si trova ad inizio del nastro e la $\delta(q_i, b) = (q_j, c, L)$

$$q_i bv \longrightarrow q_j cv$$

Non si può andare a sinistra del nastro!

Configurazione di Start di M su input w:

$q_0 w \leftrightarrow$ stato q_0 iniziale, nastro contiene w, testina sulla prima cella del nastro

Config. Accept: che raggiunge q_{accept}
Config. Reject: che raggiunge q_{reject}
Config. Halt(arresto): qualsiasi Config. Accept o Reject

2.2.3 Computazioni

Una MdT M accetta una parola w se esiste una **Computazione**(sequenza di Configurazioni) di M, $C_1, C_2, ..., C_k$ tali che

1. $C_1 = q_0$w è la Configurazione iniziale di M con input w
2. C_i produce C_{i+1} per ogni i=1,...,k-1
3. C_k è la Configurazione Accept

Tre possibile risultati di una configurazione:

1. M accetta - se si ferma in q_{accept}.
2. M rifiuta - se si ferma in q_{reject}.
3. M ciclo/loop - se non si ferma mai.

2.2.4 Linguaggio di una MdT

Linguaggio di M o Linguaggio riconosciuto da M: è l'insieme delle stringhe che accetta M
denotato con L(M)
Un Linguaggio è Linguaggio Turing riconoscibile se esiste una MdT che lo riconosce.

2.2.5 Decisore di una MdT

Un Decisore è una MdT che si ferma su ogni input(non va mai in loop).

Si chiamano Decisori perché decidono sempre l'input, per ogni input decidono se appartiene a un linguaggio. Diciamo che un decisore decide un certo Linguaggio L se riconosce tale Linguaggio L.

Un Linguaggio si dice Linguaggio Turing decidibile se esiste una MdT(che si ferma sempre) che lo decide.

2.2.6 Esempi di MdT

$L = \{0^{2^n} \mid n > 0\}$

Insieme di stringe di 0 la cui lunghezza è potenza di 2.

Linguaggio non regolare!
Vogliamo construire un MdT M_2 che lo decide.
Sia w l'input, M_2:

1. Scorro in nastro da sinistra a destra cancellando ogni **SECONDO** 0.
2. Se rimane un solo 0, **Accept**.
3. Se rimane un numero di 0 dispari $\geqslant 3$, **Reject**.
4. Riporta la testina all'inizio del nastro.
5. Go to passo 1.

n è potenza di 2 se dopo ripetute divisioni per 2 da resto 0 fino a n = 1.

$L = \{w\#w \mid w \in \{0,1\}^*\}$

Stringa w formata da 0 e 1, # e poi di nuovo la stringa w.

Idea per una possibile MdT:

- Leggi primo carattere.
- Memorizzarlo e cancellarlo.
- Cerca # e guarda il carattere successivo.
- Se sono uguali cancellali.
- Ritorna al primo carattere non cancellato.
- Ripeti fino a scorrere tutta la stringa.

su una stringa input w M_1 deve:

1. Memorizzare il simbolo più a sinistra e cancellarlo(x)
2. Scrivere sul nastro fino a superare # , se non c'è **Reject**.
3. Confronto il primo simbolo $\neq$ da # con il memorizzato.

$\Sigma = \{0,1,\#\}$ $\Gamma = \{0,1,\#,x,\_\}$ lo stato q_reject e tutte le sue transizioni sono omesse per semplicità

2.3 Alternative di MdT(Varianti)

Abbiamo delle alternative alle MdT classiche.

MdT a più nastri
MdT non deterministiche.

Tutte le varianti "ragionevoli" hanno la stessa capacità computazionale.

2.3.1 Stayer

Uno Stayer è una MdT in cui la testina può rimanere nella stessa cella del nastro.
$\sigma : Q \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$
S serve a indicare che la testina rimane ferma.

Diciamo che la potenza computazionale è la stessa per entrambi dato che riconoscono la stessa classe di linguaggi.

MdT può essere semplicimente simulato sa uno Stayer. **OVVIO!**
quindi rimane da provare che:
il potere computazionale di MdT è almeno pari a uno Stayer $\leftrightarrow$ per ogni Stayer esiste un MdT che riconosce lo stesso linguaggio.

Assumiamo che M è uno Stayer e MdT M' che simula M M' è definita come M tranne che per ogni transizione S è sostituita da 2 transizioni in M':

1. la prima porta M' in uno stato speciale(addizionale) e muove la testina a destra(R).
2. la seconda ritorna allo stato originale muovendo la testina a sinistra(L).

Quindi M e M' riconoscono lo stesso linguaggio!

Equivalenza non significa stessa velocità di esecuzione!

Quando si vuole provare l'equivalenza (stesso potere computazionale) mostriamo che possiamo simulare l'uso con l'altra.

2.3.2 Implementazione della memoria

In alcune occasioni avremo bisogno di memorizzare l'informazione letta sul nastro.

Possiamo utilizzare gli stati per memorizzare le informazioni!

Esempio: Supponiamo MdT M che deve leggere i primi 2 bit del nastro e scriverli alla fine dell'input(al posto dei 2 $\perp$ più a sinistra)

Questa è una tecnica utilizzabile in generale
Idea:

- Costruiamo MdT:
  M_{00} che legge i primi 2 bit. cerca la fine dell'input e scrive 00 alla fine dell'input
- Replichiamo per tutte le coppie possibili $M_{01}M_{10}M_{11}$
- M: dopo aver letto i primi 2 bit input, si sposta sulla replica corrispondente e scrive i bit alla fine.

Se vogliamo che M memorizzi una sequenza di k simboli possiamo:

- avere una replica per ogni possibile input di k simboli
- M si sposta sulla replica che corrisponde alla sequenza letta

Necessari circa $|\Sigma^k|$ **stati aggiuntivi**

2.3.3 MdT multi-nastro

Una MdT a k nastri è una normale MdT avente k nastri.

- Inizialmente l'input compare solo sul primo nastro e gli altri sono vuoti.
- La funzione di transizione: $\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, S, R\}^k$

  muove le testine(in modo indipendente) dei vari nastri.

Simili a MdT: usa k nastri con k $\geqslant$ 1.
Ciascun nastro ha la propria testina.

Istruzione specifica:

- k simboli letti
- k simboli da scrivere
- k movimenti delle k testine

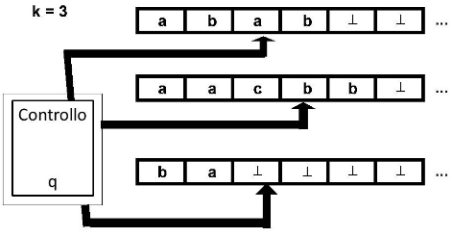


Figure 2.4: Esempio di una MdT multi-nastro

Esempio 0^n1^n con MdT multi-nastro

Possiamo implementare questo linguaggio con una MdT multi-nastro

- Il primo nastro legge gli 0 e scrive 1 sul secondo nastro e va a destra
- Quando si legge il primo 1 nel primo nastro, sul secondo nastro scorre a sinistra e controlla se hanno lo stesso numero di 1

MdT multi-nastro vs MdT classiche

Le MdT multi-nastro sembrerebbero più potenti delle MdT classiche

TH: MdT multi-nastro e MdT sono modelli equivalenti

Per provare questa equivalenza dobbiamo dimostrare il teorema in tutte e due le direzioni, ovvero che una MdT sia potente con una MdT multi-nastro e viceversa.
MdT è anche MdT multi-nastro. OVVIO! k nastri = 1
Nell'altro senso invece dobbiamo codificare le informazioni su un solo nastro

- Immagino i k nastri affiancati, ognuno con la posizione della testina indicata.
- Possiamo utilizzare il simbolo # per separare i nastri.
- ogni .... ha lunghezza variabile che dipende dal contenuto del nastro corrispondente.
- Per la testina, possiamo estendere l'alfabeto del nastro(raddoppiando i simboli) ad esempio $\Gamma = \{a, b, c\}$ e il nuovo alfabeto con $\dot{a}, \dot{b}, \dot{c}$. Le lettere con il pallino indicano che la testina si trova lì.

Se la testina S punta a un elemento del primo blocco e legge $\dot{\gamma}$: la testina del primo nastro è in questa posizione e legge γ
Alfabeto esteso:

Per ogni istruzione di MdT multi-nastro, la MdT S:

- Scorre i k nastri e "raccogli" informazioni sui k simboli letti
- Applica la transizione, scorrendo i k nastri e applicando su ciascuno scrittura e spostamento

Ora dobbiamo utilizzare il concetto di memoria!

Dimostrazione formale molto complessa!
Perché dovrebbero esistere degli stati che rappresentano tutte le configurazioni possibili(memorizzazione)

Per ogni mossa del tipo $\delta : (q, a_1, ..., a_k) = (s, b_1, ..., b_k, D_1, ..., D_7)$

- S scorre il nastro verso destra fino al primo $\perp$ memorizzando nello stato i simboli marcati sui singoli nastri.
- la testina si riposiziona all'inizio del nastro
- poi il nastro viene scorso di nuovo(eseguendo su ogni sezione (cioè un nastro di MdT) le azioni che simula quelle delle testine di M(scrittura e spostamento)
- Durante il secondo passo

Per ogni istruzione di MdT multi-nastro M, la MdT S:

1. Scorre i k nastri e "raccoglie" le informazioni sui k simboli letti
2. Applica transizione, scorrendo i k nastri e applicando su ciascuno scrittura e spostamento
3. Infine la MdT entra nello stato che ricorda il nuovo stato di S e riposiziona la testina all'inizio del nastro.

Molti più stati e mosse ma S simula M!

2.3.4 MdT non deterministica

Una MdT non deterministica NMdT è una MdT avente la funzione di transizione:

$$\delta : Q \times \Gamma \longrightarrow P(Q \times \Gamma \times \{L, R\})$$

P = insieme potenza
Analogamente alla differenza tra DFA e NFA
Consente di avere più transizione per la stessa configurazione!

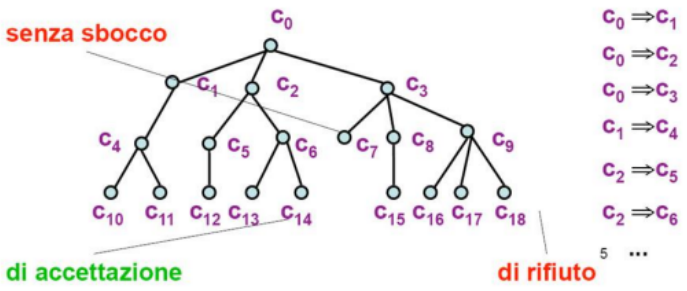


Figure 2.5: Albero MdT non deterministica

Computazione di una MdT: ben scritta da un albero contenente ogni configurazione raggiungibile dalla configurazione iniziale C_0 su un dato input. Se un cammino (da C_0) raggiunge lo stato accetta allora la NMdT accetta un input, anche se altri cammini raggiungono uno stato reject.

NMdT vs MdT classica

Anche in questo caso le NMdT sembrano più potenti di un MdT classica

In realtà sono equivalenti!

TH: NMdT e MdT sono modelli equivalenti

Lo scopo della dimostrazione è di avere una NMdT che può simulare MdT(OVVIO!) e soprattutto dimostrare che una MdT può simulare una NMdT.

Idea per dimostrare che ua MdT M può simulare una NMdT N su input w:

- M deve eseguire tutti i cammini(computazioni) di N su w
- e deve accettare se almeno una computazione raggiunto lo stato accetta

Problema!!!

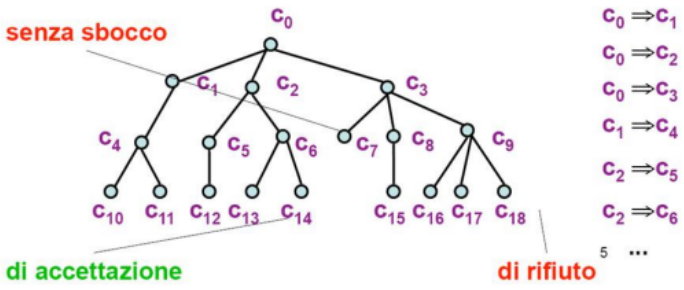
L'albero delle computazioni può andare in loop!
Un loop significa che un cammino si estende a profondità infinita, quindi nel caso intraprendiamo quel cammino rimarremmo bloccati.
Ciò corrisponde a una visita DFS sull'albero delle computazioni.

Ma noi sappiamo che se usiamo una visita BFS, ovvero visitiamo le computazioni per livello) possiamo comunque scorrere l'albero delle computazioni livello per livello fino a trovare uno stato accettante.

Simulazione fatta mediate MdT D a 3 nastri

Idea:

- Nastro 1 contiene input e non viene modificato
- Nastro 2 contiene una copia del nastro di N corrispondente a una diramazione della sua computazione non deterministica
- Nastro 3 serve a tenere traccia della posizione di D nell'albero della computazione non deterministica di N.



Nastro 3: contiene 222, significa che D sta simulando la configurazione C_{14} di N, perché significa che partendo dalla radice scorriamo l'albero di figlio in figlio.
b = 3 $\longrightarrow$ max numero di figli quindi sul nastro 3 ci saranno stringhe su $\{1, 2, 3\}$

2.4 Funzioni calcolabili

Teorema:
Una funzione f è calcolabile se esiste una MdT M tale che la configurazione iniziale è $q_0 w$ e calcolando $f(w)$ si ha una configurazione di accept, per ogni $w \in D$ (D = dominio)

Chapter 3

Decidibilità

3.1 Problemi di decisione

I problemi di decisione sono problemi che hanno come soluzione una risposta SI o NO.
Rappresenteremo i problemi di decisione mediante linguaggi.

Es: il linguaggio che rappresenta il problema "PRIMO" è

$$P = \{\langle x \rangle \mid x \text{ è un nuemro primo}\}$$

dove $\langle x \rangle$ denota una "ragionevole" codifica di x mediante una stringa su un alfabeto Σ .

qui parliamo della rappresentazione dell'oggetto x, tra parentesi angolari vogliamo dire che l'oggetto è rappresentato tramite una stringa. Risolvere PRIMO equivale a decidere il linguaggio P.
In questo modo esprimiamo un problema computazionale come un problema di riconoscimento di un linguaggio(insieme delle codifiche di istanze SI per il problema).

Altro Esempio:
Sia $A = \{\langle G \rangle \mid G$ è un grafo connesso $\}$ un linguaggio di stringhe che rappresentano grafi connessi(non orientati)
$\langle G \rangle \in A$ se e solo se G è istanza SI per CONNESSO
Risolvere CONNESSO equivale a decidere il linguaggio A

In questo modo esprimiamo un problema computazionale come un problema di riconoscimento di un linguaggio (insieme delle codifiche di istanze SI per il problema)

Obiettivo: analizzare i limiti della risoluzione dei problemi mediante algoritmi.

Studieremo il potere computazionale degli algoritmi nella soluzione dei problemi.

Proveremo che esistono problemi che possono essere risolti medianti algoritmi e altri no. In pratica esistono dei problemi per cui non è possibile creare un algoritmo di risoluzione. Quindi vedremo che ci sono problemi che non possono essere decisi da una MdT.

3.2 Problemi indecidibili

Obiettivo: presentare un problema irrisolvibile

Problema
Input: MdT M, stringa w
Domanda: M si arresta su input w?

Linguaggio corrispondente

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ è una MdT che si arresta su } w\}$$

Teorema
Il linguaggio $HALT_{TM}$ non è decidibile.

Proveremo che per questo linguaggio non esiste un decider, quindi una MdT che decide questo linguaggio.

3.2.1 Strumenti

Cardinalità di insiemi (infiniti).

Diagonalizzazione: metodo introdotto da Cantor.

Cardinalità

Cardinalità di un insieme: la sua taglia

Due insiemi hanno la stessa cardinalità se è possibile stabilire una corrispondenza tra i loro elementi.

Es: $A = \{1, 2, 3\}, B = \{4, 3, 5\} \quad \Rightarrow \quad 1 - 4, 2 - 3, 3 - 5$

Cosa possiamo dire per insiemi infiniti?

Paradosso di Hilbert

Per capire meglio la cosa, prima di andare avanti, andiamo a vedere quello che è il paradosso di Hilbert (matematico tedesco che studiava gli insiemi)

Nel paese senza confini esiste il più grande di tutti gli alberghi: l'albergo con le stanze infinite. Tuttavia anche gli ospiti sono infiniti, e il proprietario ha esposto un cartello con la scritta COMPLETO.
Ad un trattto si presenta un viaggiatore che ha assolutamente bisogno di una camera per la notte. Egli non fa questione di prezzo e infine convince l'albergatore, il quale trova modo di alloggiarlo.
Come fa?

numero stanze = numero ospiti

A ogni ospite si chiede di spostarsi nella stanza successiva, perché noi sappiamo che per ogni intero c'è un successivo e sappiamo che abbiamo infinite stanze, quindi ora la stanza 1 sarà libera.

numero stanze = numero ospiti + 1 Paradosso!

Poco dopo arriva una comitiva di **infiniti turisti**, anche in questo caso l'albergatore si lascia convincere, in fondo si tratta di un grosso affare, e trova posto ai **nuovi infiniti ospiti** con la stessa facilità con cui aveva alloggiato l'ospite in più.
Come fa(senza ripetere infinite volte il passo visto prima)?

A ogni ospite si chiede di spostarsi al numero di stanza doppio rispetto la propria stanza ,lasciando ai nuovi ospiti tutte le camere con i numeri dispari, che sono essi stessi infiniti, risolvendo il problema.
Gli ospiti sono dunque tutti sistemati, anche se l'albergo fosse pieno.

numero stanze = numero ospiti + infinita comitiva Paradosso!

Un altro problema per il nostro albergatore: ci sono infiniti alberghi con infinite stanze tutti al completo. Tutti gli alberghi chiudono, tranne quello del nostro albergatore che ora si ritrova invaso dagli ospiti che sono in cerca di stanze.
Come fa(senza ripetere infinite volte il passo visto prima)?

A ogni ospite si assegna una coppia di numeri (n, m) in cui n indica l'albergo di provenienza, e m la relativa stanza dell'albergo da cui proveniva. Gli ospiti sono quindi etichettati in questo modo:

$$\begin{array}{cccccc} (1,1) & (1,2) & \cdots & (1,m) & \cdots \\ (2,1) & (2,2) & \cdots & (2,m) & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ (n,1) & (n,2) & \cdots & (n,m) & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{array}$$

A questo punto basta assegnare le nuove stanze agli ospiti secondo un criterio ordinato, ad esempio per diagonali:
(1,1) $\to$ 1; (1,2) $\to$ 2; (2,1) $\to$ 3; (1,3) $\to$ 4; (2,2) $\to$ 5; (3,1) $\to$ 6; ...

Metodo della diagonalizzazione di Cantor

Introdotto da Cantor nel 1973 mentre cercava di determinare come stabilire se, dati due insiemi infiniti, uno è più grande dell'altro.

- Cantor osservò che due insiemi finiti hanno la stessa cardinalità se gli elementi dell'uno possono essere messi in corrispondenza uno a uno con quelli dell'altro(bietttiva)

Definizione
Due insiemi X e Y hanno la stessa cardinalità se esiste una funzione biettiva f: $X \to Y$ di X su Y.

$$|X| = |Y| \Leftrightarrow \text{esiste una funzione biettiva } f : X \to Y$$

ESEMPIO $f : \{1, 2, 3\} \to \{2, 4, 7\}$ $\quad 1 \to 2, 2 \to 4, 5 \to 7$ INSIEME FINITO
ESEMPIO $f : \mathbb{N} \to \{2n \mid n \in \mathbb{N}\}$ $\quad n \to 2n$ INSIEME INFINITO

Così stiamo dicendo che l'insieme degli interi e l'insieme dei numeri pari hanno la stessa cardinalità tramite la funzione biettiva.

Insiemi numerabili

Definizione
Un insieme è enumerabile se ha la stessa cardinalità di un sottoinsieme di $\mathbb{N}$.

Se A è numerabile possiamo "numerare" gli elementi di A e scrivere una lista $(a_1, a_2, ...)$
cioè per ogni numero naturale i, possiamo specificare l'elemento $i - mo$ della lista.
Es. Per l'insieme dei numeri naturali pari, l'elemento $i - mo$ della lista corrisponde a $2i$.

3.2.2 Metodo diagonalizzazione

$$\{w_1, w_2, w_3, ...\} = \Sigma^*, \{M_1, M_2, M_3, ...\} = \{ \text{ MdT su } \Sigma\} : \text{numerabili}$$

Σ^* è numerabile quindi possiamo numerare tutte le stringhe.
Tutte le macchine di Turing M possono essere numerate.

Se possiamo numerare sia le stringhe che le macchine di Turing possiamo costruire una tabella.

	w_1	w_2	w_3	...	w_i	w_j
M_1	$x_{1,1}$	.	.	.	.	.
M_2	.	$x_{2,2}$	.	.	.	.
.	.	.	$x_{3,3}$	.	.	.
.	.	.	.	$x_{4,4}$	.	.
M_i	.	.	.	.	$x_{i,i}$	$x_{i,j}$
.	.	.	.	.	.	.
.	.	.	.	.	.	.

L'i-esima riga corrisponde all'i-esima MdT.
L'i-esima colonna corrisponde all'i-esima stringa w.
Indichiamo con $x_{i,j} = 1$ se $w_j \in L(M_i)$, 0 altrimenti.

$$x_{1,1} = \left\{ \begin{array}{lll} 1 & \text{se} & w_1 \in L(M_1) \\ 0 & \text{se} & w_1 \notin L(M_1) \end{array} \right\}$$

Sia $L = \{w_i \in \Sigma^* \mid w_i \notin L(M_i)\}$
Quindi L è il "complemento della diagonale":
se l'elemento (M_i, w_i) della diagonale è $x_{i,i} = 1$, allora $w_i \notin L$
se l'elemento (M_i, w_i) della diagonale è $x_{i,i} = 0$, allora $w_i \in L$

Quindi ora abbiamo un linguaggio L che per definizione potrebbe essere rappresentato da una MdT in lista.
Supponiamo che L sia in lista.
se $w_i \in L \Rightarrow x_{i,i} = 0 \Rightarrow w_i \notin L(M_i) = L$ contraddizione!
se $w_i \notin L \Rightarrow x_{i,i} = 1 \Rightarrow w_i \in L(M_i) = L$ contraddizione!

Da queste contraddizioni deduciamo che esiste almeno un linguaggio che non è riconosciuto da una MdT.

"**Quindi esistono più linguaggi che MdT!!!**"

3.3 MdT universale

Una MdT Universale U simula la computazione di una qualsiasi MdT M. Sarà una MdT capace di simulare una qualsiasi altra MdT su ogni input.

U riceve in input una descrizione $\langle M, w\rangle$ di M e di un input associato a essa.

$$\langle M, w\rangle \rightarrow \boxed{\text{Macchina di Turing Universale U}} \rightarrow \left\{ \begin{array}{lll} accetta & \text{se} & M \text{ accetta } w \\ rifiuta & \text{se} & M \text{ rifiuta } w \end{array} \right\}$$

M sta a indicare una qualsiasi MdT e w un qualsiasi input che possiamo dare a M.

3.3.1 MdT che simula un'altra MdT

Sappiamo che per costruzione una MdT può simulare un automa, ma si può dimostrare che una MdT può simulare anche un'altra MdT.

Come?
Procedimento:

1. Marca stato iniziale di M (stato corrente) e primo simbolo su nastro (posizione corrente testina)
2. Cerca prossima transizione (nella parte che descrive la funzione di transizione), sia (q, x) $\rightarrow (q', x'$, D)
3. Esegui la transizione
4. Aggiorna lo stato corrente (marca q') e la posizione corrente della testina (marca simbolo a D)
5. Se lo stato corrente risulta q_{accept}/q_{reject} decidi di conseguenza, altrimenti ripeti da 2

3.4 A_{TM} è Turing riconoscibile

Teorema:
Il linguaggio $A_{TM} = \{\langle M, w\rangle \mid$ M è una MdT che accetta la parola w $\}$ *è Turing riconoscibile*

Dimostrazione
Definiamo una MdT U che accetta A_{TM} sull'input $\langle M, w\rangle$, dove M è una MdT e w è una stringa

1. Simula M sull'input w.
2. Se M accetta, accetta; se M rifiuta, rifiuta.

Nota: U è detta MdT Universale
Nota: U riconosce A_{TM}; accetta ogni coppia $\langle M, w\rangle \in A_{TM}$
Nota bene: U **cicla su** $\langle M, w\rangle$ **se e solo se** M **cicla su** w**. Quindi** A_{TM} **è riconoscibile ma non decidibile.**

3.5 A_{TM} non è decidibile

Teorema:
Il linguaggio $A_{TM} = \{\langle M, w\rangle \mid$ M è una MdT che accetta la parola w $\}$ *non è decidibile.*

Dimostrazione:
Supponiamo per assurdo che esiste una MdT H che è un decider quindi avrà solo due possibili risultati ovvero accetta o rifiuta.

$$H = \left\{ \begin{array}{lll} accetta & \text{se} & M \text{ accetta } w \\ rifiuta & \text{se} & M \text{ non accetta } w \end{array} \right\}$$

Adesso costruiamo un a nuova nuava MdT D che usa H come sotto programma di D sull'input $\langle M\rangle$, dove M è una MdT:

1. Simula H sull'input $\langle M, \langle M\rangle\rangle$
2. Come output fornisce l'inverso di H, cioè se H accetta, *rifiuta* e se H rifiuta, *accetta*.

$$\langle M\rangle \rightarrow \boxed{\text{D} \rightarrow \langle M, \langle M\rangle\rangle \rightarrow \text{H} \rightarrow \left\{ \begin{array}{l} accetta \\ rifiuta \end{array} \right\} \rightarrow \text{I}} \rightarrow \left\{ \begin{array}{l} rifiuta \\ accetta \end{array} \right\}$$

Quindi

$$D(\langle M\rangle) = \left\{ \begin{array}{lll} rifiuta & \text{se} & M \text{ accetta } \langle M\rangle \\ accetta & \text{se} & M \text{ non accetta } \langle M\rangle \end{array} \right\}$$

Se ora diamo in pasto a D la sua stessa descrizione ($\langle D\rangle$ abbiamo

$$D(\langle D\rangle) = \left\{ \begin{array}{lll} rifiuta & \text{se} & D \text{ accetta } \langle D\rangle \\ accetta & \text{se} & D \text{ non accetta } \langle D\rangle \end{array} \right\}$$

Cioè D accetta $\langle D\rangle$ se e solo se non accetta $\langle D\rangle$.

ASSURDO!

Tutto ciò è causato che abbiamo assunto H esiste come decider.

Quindi H **non esiste!** .

3.5.1 A_{TM} e la tecnica della diagonalizzazione

Rappresentiamo il tutto con una bella tabella: come indice di ogni riga mettiamo la MdT M_i, e come indice delle colonne la stringa $\langle M_j \rangle$.
Se una M_i accetta una stringa $\langle M_j \rangle$, scriveremo accetta nella cella (i,j), rifiuta altrimenti.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

Aggiungiamo poi un'ultima riga per il decisore D, e un'ultima colonna per la stringa $\langle D \rangle$ che lo descrive. La riga D è riempita scrivendo l'opposto delle celle che si trovano sulla diagonale della stessa colonna.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$			$\vdots$				$\ddots$

Cosa scriviamo nella cella che incrocia D e $\langle D \rangle$?
Contraddizione poiché l'elemento in posizione (D,$\langle D \rangle$) deve essere l'opposto di se stesso!

3.6 Un linguaggio non Turing riconoscibile

Un linguaggio L è co-Turing riconoscibile se $\overline{L}$ è Turing riconoscibile.

Teorema: Un linguaggio L è decidibile se e solo se L è Turing riconoscibile e co-Turing riconoscibile.

L è decidibile $\Leftrightarrow$ L e il suo complemento sono entrambi Turing riconoscibili.

Dimostrazione
($\Rightarrow$) Se L è decidibile allora esiste una macchina di Turing M con due possibili risultati di una computazione (accettazione, rifiuto) e tale che M accetta w se e solo se w $\in$ L. Allora L è Turing riconoscibile. Inoltre è facile costruire una MdT M che accetta w se e solo se w $\notin$ L:

$$w \to \boxed{M} \to \left\{ \begin{array}{llll} accetta & \text{se} & w \in L \\ rifiuta & \text{se} & w \notin L \end{array} \right\} \to \left\{ \begin{array}{llll} rifiuta & \text{se} & M \text{ accetta} \\ accetta & \text{se} & M \text{ non accetta} \end{array} \right\}$$

($\Leftarrow$) Supponiamo che L e il suo complemento siano entrambi Turing riconoscibili.

Sia M_1 una MdT che riconosce L e M_2 una MdT che riconosce $\overline{L}$.
Definiamo una MdT N (a due nastri): sull'input x

1. Copia x sui nastri di M_1 e M_2
2. Simula M_1 e M_2 in parallelo (usa un nastro per M_1, l'altro per M_2
3. Se M_1 accetta, accetta; se M_2 accetta, rifiuta

$$x \to \boxed{\boxed{M_1} \to \text{accetta} \quad \boxed{M_2} \to \text{accetta}} \to \left\{ \begin{array}{llll} accetta & \text{se} & M_1 \text{ accetta} \\ rifiuta & \text{se} & M_2 \text{ accetta} \end{array} \right\}$$

N decide L. Infatti, per ogni stringa x abbiamo due casi:

1. x $\in$ L. Ma x $\in$ L se e solo se M_1 si arresta e accetta x. Quindi N accetta x.
2. x $\notin$ L. Ma $\notin$ L se e solo se M_2 si arresta e accetta x. Quindi N rifiuta x.

Poiché una e solo una delle due MdT accetta x, N è una MdT con solo due possibili risultati di una computazione (accettazione, rifiuto) e tale che N accetta x se e solo se x $\in$ L.

3.6.1 $\overline{A_{TM}}$ non è Turing riconoscibile

$\overline{A_{TM}}$ non è Turing riconoscibile

Dimostrazione

Supponiamo per assurdo che $\overline{A_{TM}}$ sia Turing riconoscibile. Inoltre, secondo il teorema visto in precedenza che A_{TM} è Turing riconoscibile. Quindi in questo caso A_{TM} risulta sia Turing riconoscibile che co-Turing riconoscibile, in conclusione risulta che A_{TM} è decidibile. Tutto ciò è un assurdo in quanto è stato già dimostrato che A_{TM} è non decidibile.

Chapter 4

Riducibilità

4.1 Cosa si intende per riducibilità?

La riducibilità è un metodo per convertire l'istanza di un problema A in un'istanza di un problema B e utilizzare la soluzione di quest'ultimo per ottenere la soluzione di A.
La riducibilità è anche uno dei metodi principali per dimostrare che un problema non è computazionalmente decidibile.

Quando A è riducibile a B allora:

- risolvere A non può essere più difficile di risolvere B
- risolvere B ci permette di ottenere la soluzione di A
- se B è decidibile allora lo è anche A
- se A non è decidibile allora anche B non è decidibile

l'idea e che se noi sappiamo per un problema P' è stato dimostrato l'indicibilità, possiamo dimostrare che P' non è più difficile di P.

Esempio: $\Sigma = \{0, 1\}$

$EVEN = \{w \in \Sigma^* \mid w$ è la rappresentazione binaria di $n \in \mathbb{N}$ pari $\}$

$ODD = \{w \in \Sigma^* \mid w$ è la rappresentazione binaria di $n \in \mathbb{N}$ dispari $\}$

Vogliamo vedere che un problema P' non è più difficile del problema P.
Possiamo costruire una MdT che effettua l'incremento.
$w \to \boxed{\textit{INCR}} \to w'$ (rappresentazione binaria di $n + 1$)

in definitiva *EVEN* non è più facile di *ODD* se esiste una MdT R che decide *ODD*, la MdT S decide *EVEN*.

$$\text{S: } w \to \boxed{\textit{INCR}} \to w' \to \boxed{\text{R}}$$

4.2 Il vero problema della fermata $HALT_{TM}$

$HALT_{TM} = \{\langle M, w\rangle \mid M$ è una MdT e M si arresta su $w\}$

Se $HALT_{TM}$ fosse decidibile potremmo decidere anche A_{TM}:
Sia R una MdT che decide $HALT_{TM}$.
Costruiamo S (che decide A_{TM}) che sull'input $\langle M, w\rangle$, dove M è una MdT e w è una stringa:

- S simula R su $\langle M, w\rangle$
- se R rifiuta, allora S rifiuta (poiché M va in loop $w \notin L(M)$);
- se R accetta, (questo significa che M si ferma su w) allora simula M finché M si ferma su w.

Se esistesse R che decide $HALT_{TM}$ allora otterremmo S che decide A_{TM}. Ma poiché sappiamo che A_{TM} è indecidibile allora R non può esistere e $HALT_{TM}$ deve essere indecidibile.

4.3 Schema di riduzione

Dal problema A al problema B

1. Sappiamo che A risulta indecidibile
2. Vogliamo provare che B è indecidibile
3. Assumiamo per assurdo B decidibile ed usiamo questa assunzione per provare A decidibile
4. La contraddizione ci fa concludere che B è indecidibile

4.4 Problema del vuoto E_{TM}

$$E_{TM} = \{\langle M \rangle \mid M \text{ MdT tale che } L(M) = \emptyset\}$$

Teorema: E_{TM} *è indecidibile*

1. Sappiamo che A_{TM} è indecidibile
2. Vogliamo provare che E_{TM} è indecidibile
3. Assumiamo per assurdo E_{TM} decidibile e proviamo che A_{TM} sia decidibile
4. La contraddizione ci fa concludere che E_{TM} sia indecidibile

Assumiamo che E_{TM} sia decidibile e mostriamo che anche A_{TM} lo è.
Sia R una MdT che decide E_{TM}, utilizziamo R per costruire una MdT S che decide A_{TM}.

Dimostrazione:
L'idea sarebbe di far eseguire ad S l'output di R su input $\langle M \rangle$. Se $L(M) = \emptyset$ quindi M non accetterà w. Ma se R rifiuta $\langle M \rangle$ quello che sappiamo è che il linguaggio di M non è vuoto quindi può accettare qualche stringa.
Il problema è che non sappiamo se M accetta la stringa w in particolare, quindi abbiamo bisogno di una piccola modifica di M.
Modifichiamo M in M_1 tale che su input x M_1 possa accettare solo w, se non accetta w vuol dire che $L(M_1) = \emptyset$.

$$L(M_1) = \begin{cases} \{w\} & \text{se } M \text{ accetta } w \\ \phi & \text{se } M \text{ rifiuta } w \end{cases}$$

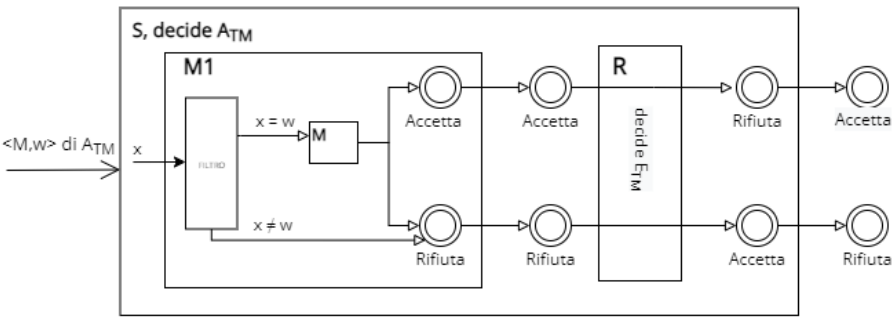
Dando in input $\langle M_1 \rangle$ ad R si avrà che:

- se R accetta vuol dire che M_1 ha rifiutato (caso in cui $L(M_1) = \emptyset$);
- se R non accetta vuol dire che M_1 ha accettato (caso in cui $L(M_1) = \{w\}$).

Possiamo utilizzare tutto questo ragionamento per costruire S(decisore di A_{TM}).

- Se R accetta S rifiuta quindi $w \notin L(M)$;
- se R rifiuta S accetta perché $L(M_1) = \{w\}$ e $w \in L(M)$

Tutto ciò è un assurdo poiché abbiamo costruito un decider S per A_{TM} che sappiamo essere non decidibile.



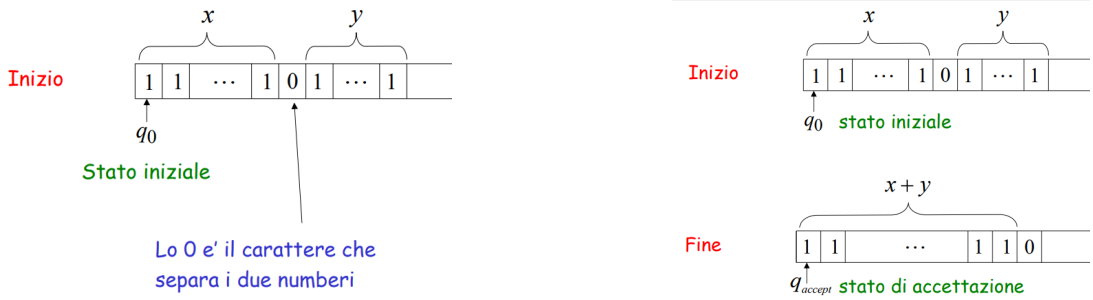
4.5 Funzioni calcolabili

Definizione:
Una funzione f si dice calcolabile se : $\Sigma^* \to \Sigma^*$ è calcolabile se esiste una MdT M tale che su ogni input w, M si arresta con f(w), e solo con f (w), sul suo nastro.

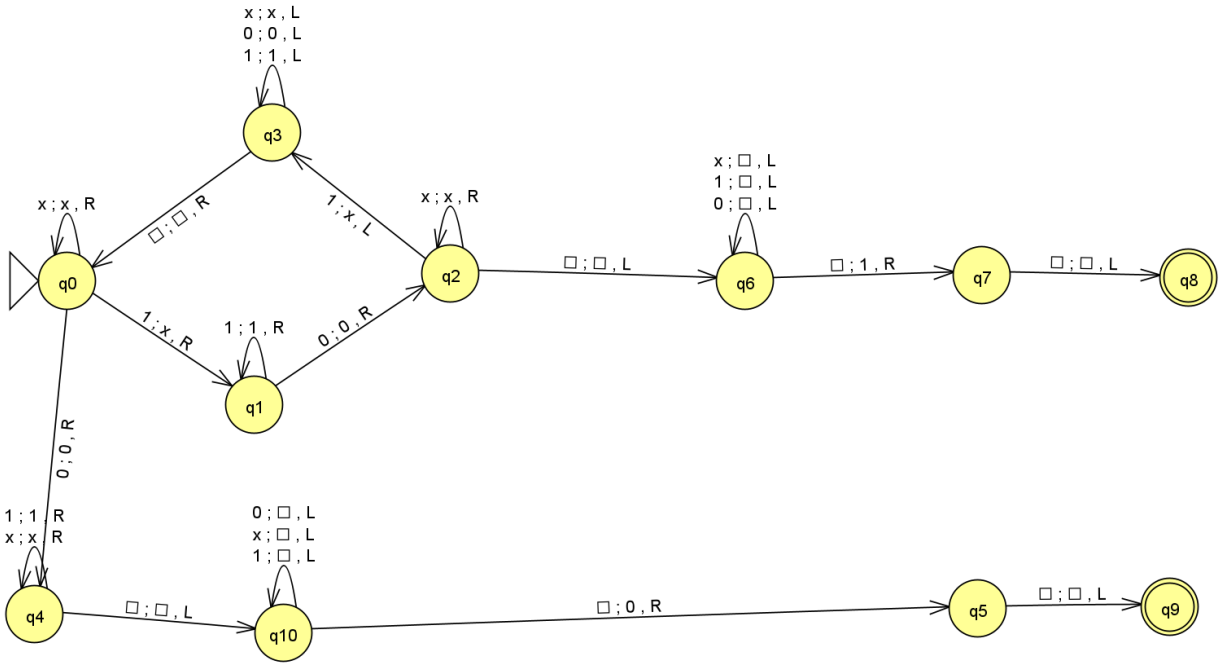
Esempio di funzione calcolabile

La funzione addizione $f(x, y) = x + y$

Per semplicità si utilizza la rappresentazione dei numeri in unario.



Un altro esempio potrebbe essere la funzione che $f(x, y) = \left\{ \begin{array}{ll} 1 & \text{se } x > y \\ 0 & \text{se } x \leq y \end{array} \right\}$



N.B: Questa MdT è stata ideata da MDR & Luca Boffa, utilizzare con cautela. Dovrebbe funzionare

Altri esempi di funzioni aritmetiche calcolabili (dove n,m $\in \mathbb{N}$)

- $incr(n) = n + 1$
- $(m, n) = m - n$
- $decr(n) = n - 1,\ se\ n > 0;\ altrimenti\ 0,\ se\ n = 0$

4.6 Riducibilità, definizione formale

Definizione:
Un linguaggio A è riducibile a un linguaggio B $(A \leq_m B)$ *se esiste una funzione calcolabile* $f : \Sigma^* \to \Sigma^*$ *tale che* $\forall w$

$$w \in A \Leftrightarrow f(w) \in B$$

La funzione f è chiamata la **riduzione di A a B**.

Teorema:

Se $(A \leq_m B)$ *e B è decidibile, allora A è decidibile.*

Siano: M decider di B e f la riduzione da A a B.
Costruiamo un decider N per A tale che su input w:

- N calcola f(w);
- N utilizza M su f(w) e dà lo stesso output.

$w \in A \Leftrightarrow f(w) \in B$ (f funzione di riduzione da A a B) $\Leftrightarrow$ M accetta $f(w)$. Quindi N **decide** A.

Teorema:

Se $(A \leq_m B)$ *e B è Turing-riconoscibile, allora A è Turing-riconoscibile.*

Siano: R_B MdT che riconosce B e f la riduzione da A a B.
Costruiamo una MdT R_A che riconosce A tale che su input w:

- R_A calcola f(w);
- R_A utilizza R_B su f(w) e dà lo stesso output.

$w \in A \Leftrightarrow f(w) \in B$ (f funzione di riduzione da A a B) $\Leftrightarrow$ R_B accetta $f(w)$. Quindi R_A **riconosce** A.

Corollario importantissimo
Se $(A \leq_m B)$ *e B è indecidibile, allora A è indecidibile.*
Se B fosse decidibile per il teorema precedente anche lo sarebbe, quindi si tratta di una contraddizione!

Corollario importantissimo, ma non troppo
Se $(A \leq_m B)$ *e B non è Turing-riconoscibile, allora A è Turing-riconoscibile.*

4.7 $A_{TM} \leq_m \overline{E}_{TM}$

$$A_{TM} = \{\langle M, w\rangle \mid \textit{ M è una MdT che accetta la parola w }\}$$

$$E_{TM} = \{\langle M\rangle \mid M \textit{ MdT tale che } L(M) = \emptyset\}$$

Facciamo vedere che esiste la riduzione $A_{TM} \leq_m \overline{E}_{TM}$

$\exists f \quad f(\langle M, w\rangle) \mid \langle M, w\rangle \in A_{TM} \Leftrightarrow f(\langle M, w\rangle) \in \overline{E}_{TM}$

Consideriamo $f : \Sigma^* \longrightarrow \Sigma^* \mid f(\langle M, w\rangle) = \langle M_1\rangle$ dove M_1 su input x:

1. Se $x \neq w$ M_1 si ferma e rifiuta x
2. Se $x = w$ M_1 simula M su w e accetta x se M accetta w

f è una riduzione di A_{TM} a $\overline{E}_{TM}$

Dimostrazione: la funzione f è calcolabile

M accetta w (cioè $\langle M, w\rangle \in A_{TM}$) sse $L(M_1) \neq \emptyset$ (cioè sse $\langle M_1\rangle \in \overline{E}_{TM}$)

$$L(M_1) = \begin{cases} \{w\} & \text{se } M \text{ accetta } w \\ \phi & \text{se } M \text{ rifiuta } w \end{cases}$$

Quindi possiamo dire che abbiamo 2 casi:

- $\langle M, w\rangle \in A_{TM} \Rightarrow L(M_1) = \{w\} \Rightarrow \langle M_1\rangle \in \overline{E}_{TM} \Rightarrow f(\langle M, w\rangle) \in \overline{E}_{TM}$
- $\langle M, w\rangle \notin A_{TM} \Rightarrow L(M_1) = \emptyset \Rightarrow \langle M_1\rangle \notin \overline{E}_{TM} \Rightarrow f(\langle M, w\rangle) \notin \overline{E}_{TM}$

Sapendo che A_{TM} è indecidibile $\Rightarrow \overline{E}_{TM}$ indecidibile

4.8 $A_{TM} \leq_m REGULAR_{TM}$

$$REGULAR_{TM} = \{\langle M\rangle \mid \textit{ M è una MdT e } L(M) \textit{ è regolare }\}$$

Facciamo vedere che esiste la riduzione $A_{TM} \leq_m REGULAR_{TM}$

$\exists f$ calcolabile $\mid \forall \langle M, w\rangle$
$\langle M, w\rangle \in A_{TM} \Leftrightarrow f(\langle M, w\rangle) \in REGULAR_{TM}$

Consideriamo $f : \langle M, w\rangle \longrightarrow \langle R\rangle$ è una riduzione da A_{TM} a $REGULAR_{TM}$, dove R su input x:

1. Se $x \in \{0^n1^n \mid n \in \mathbb{N}\}$, allora R si ferma e accetta x
2. Se $x \notin \{0^n1^n \mid n \in \mathbb{N}\}$, allora R simula M su w e accetta x se M accetta W

$$L(R) = \left\{ \begin{array}{llll} x = 0^n1^n & ACCETTA & & \\ x \neq 0^n1^n & ACCETTA & \text{se} & \langle M, w\rangle \in A_{TM} \\ x \neq 0^n1^n & RIFIUTA & \text{se} & \langle M, w\rangle \notin A_{TM} \end{array} \right\}$$

Praticamente costruiamo R in modo che se accetta abbiamo un linguaggio che è uguale a Σ^* e se rifiuta abbiamo un linguaggio non regolare ($\{0^n1^n\}$), così facendo creando la funzione di riduzione che definisce R possiamo ridurre il problema a A_{TM}.

Quindi possiamo dire che abbiamo 2 casi:

- $\langle M, w\rangle \in A_{TM} \Rightarrow L(R) = \Sigma^*$ è regolare $\Rightarrow \langle R\rangle \in REGULAR_{TM}$
- $\langle M, w\rangle \notin A_{TM} \Rightarrow L(R) = \{0^n1^n\}$ non è regolare $\Rightarrow \langle R\rangle \notin REGULAR_{TM}$

4.9 $E_{TM} \leq_m EQ_{TM}$

$$E_{TM} = \{\langle M \rangle \mid M \textit{ MdT tale che } L(M) = \emptyset\}$$

$$EQ_{TM} = \{\langle M_1,\ M_2 \rangle \mid M_1,\ M_2 \textit{ sono MdT tale e } L(M_1) = L(M_2)\}$$

Facciamo vedere che esiste la riduzione $E_{TM} \leq_m EQ_{TM}$

$\exists f$ calcolabile $\quad | \quad \forall \langle M \rangle$
$\langle M \rangle \in E_{TM} \Leftrightarrow f(\langle M \rangle) \in EQ_{TM}$

Consideriamo M_1 una MdT tale che L(M_1)=$\emptyset$ e sia $f : \langle M \rangle \longrightarrow \langle M,\ M_1 \rangle$ riduzione da E_{TM} a EQ_{TM}.

1. $\langle M \rangle \in E_{TM} \Rightarrow$
   per definizione di E_{TM} avremo che $L(M) = \emptyset$, per definizione avremo anche che $L(M_1) = \emptyset$.
   Il valore della funzione $f(\langle M \rangle) = \langle M,\ M_1 \rangle$ i quali linguaggi sappiamo essere entrambi $\emptyset$ quindi uguali, otterremo che $\langle M,\ M_1 \rangle \in EQ_{TM}$.

2. $\langle M \rangle \notin E_{TM} \Rightarrow$
   per definizione di E_{TM} avremo che $L(M) \neq \emptyset$, per definizione avremo anche che $L(M_1) = \emptyset$.
   Il valore della funzione $f(\langle M \rangle) = \langle M,\ M_1 \rangle$ i quali linguaggi sappiamo essere rispettivamente $L(M) \neq \emptyset$ e $L(M_1) = \emptyset$ quindi diversi, otterremo che $\langle M,\ M_1 \rangle \notin EQ_{TM}$.

Tale funzione descritta ci dà una riduzione da E_{TM} a EQ_{TM}

4.10 Teorema di Rice, detto Teorema del Riso

$$\textit{Sia } L_{\mathcal{P}} = \{\langle M \rangle \mid M \textit{ è una MdT che verifica la proprietà } \mathcal{P}\}$$

un linguaggio che verifichi le seguenti 2 proprietà:

1. L'appartenenza di M a $\mathcal{P}$ dipende solo da $L(M)$, ovvero

   $$\forall M_1, M_2 \text{ MdT tali che } L(M_1) = L(M_2), \langle M_1 \rangle \in L_{\mathcal{P}} \Leftrightarrow \langle M_2 \rangle \in L_{\mathcal{P}}$$

2. $\mathcal{P}$ è una proprietà non banale se

   $$\exists M_1, M_2 \text{ MdT tali che } \langle M_1 \rangle \in L_{\mathcal{P}} \text{ e } \langle M_2 \rangle \notin L_{\mathcal{P}}$$

Se il teorema è soddisfatto $L_{\mathcal{P}}$ è indecidibile

NOTA BENE: Per proprietà **non banale** si intende in poche parole che esiste almeno una macchina di Turing che soddisfa la proprietà $\mathcal{P}$ e almeno una macchina di Turing che non soddisfa la proprietà $\mathcal{P}$.

CONCLUSIONI AFFRETTATE POST-RICE
Non possiamo decidere se una MdT:

- Accetta $\emptyset$
- Accetta un linguaggio finito.
- Accetta un linguaggio regolare, ecc

4.10.1 Teorema di Rice su input pari(accetta)

$$L = \{\langle M \rangle \mid M \textit{ è una MdT che accetta ogni input di lunghezza pari }\}$$

Come prima cosa verifichiamo che la proprietà $\mathcal{P}$ non è banale, cioè che:

$$\exists M_1, M_2 \text{ MdT tali che } \langle M_1 \rangle \in L_{\mathcal{P}} \text{ e } \langle M_2 \rangle \notin L_{\mathcal{P}}$$

Quindi nel nostro caso possiamo dire che esistono due MdT tali che il linguaggio $L(M_1) = \Sigma^* \in L$ e il linguaggio $L(M_2) = \emptyset \notin L$

Proprità non banale!

Ora dobbiamo verificare l'altra proprietà del teorema:

$$\forall M_1, M_2 \text{ MdT tali che } L(M_1) = L(M_2), \langle M_1 \rangle \in L_{\mathcal{P}} \Leftrightarrow \langle M_2 \rangle \in L_{\mathcal{P}}$$

Possiamo dire che per ogni MdT M_1 e M_2 tali che $\langle M_1 \rangle \in L \Leftrightarrow L(M_1) = L(M_2) \Leftrightarrow \langle M_2 \rangle \in L$

Visto che sono soddisfatte le due proprietà del Teorema di Rice possiamo affermare che L è indecidibile

4.10.2 Teorema di Rice su input dispari(arresta)

$$L = \{\langle M \rangle \mid M \textit{ è una MdT che si arresta su ogni input di lunghezza dispari }\}$$

Come prima cosa verifichiamo che la proprietà $\mathcal{P}$ non è banale, cioè che:

$$\exists M_1, M_2 \text{ MdT tali che } \langle M_1 \rangle \in L_{\mathcal{P}} \text{ e } \langle M_2 \rangle \notin L_{\mathcal{P}}$$

Quindi nel nostro caso possiamo dire che esistono due MdT tali che il linguaggio $L(M_1) = \Sigma^* \in L$ e il linguaggio $L(M_2) = \emptyset \notin L$

Proprità non banale!

Ora dobbiamo verificare l'altra proprietà del teorema:

$$\forall M_1, M_2 \text{ MdT tali che } L(M_1) = L(M_2), \langle M_1 \rangle \in L_{\mathcal{P}} \Leftrightarrow \langle M_2 \rangle \in L_{\mathcal{P}}$$

Ora abbiamo il problema di dimostrare che le due MdT si arrestano su ogni input dispari, ma sappiamo che due macchine possono riconoscere lo stesso linguaggio ma una si arresta su ogni input(decider) e l'altra no.

$$\exists M_1, M_2 \mid L(M_1) = L(M_2) = A \text{ con } \langle M_1 \rangle \in L \text{ e } \langle M_2 \rangle \notin L$$

$\exists\, w \notin A \mid M_1$ rifiuta (decider) quindi $\langle M_1 \rangle \in A$ e M_2 va in loop(riconoscitore), per definizione $\langle M_2 \rangle \notin A$.

Visto che una delle due proprietà non è soddisfatta il Teorema di Rice sul linguaggio L non è applicabile

4.10.3 Teorema di Rice su input $\langle M, w \rangle$(non si applica)

$$L = \{\langle M,\ w \rangle \mid M \textit{ è una MdT e w non appartiene a L(M)}\}$$

Rice in questo caso non è applicabile perchè il teorema può essere applicato solo su codifiche, $\langle M \rangle$ di Macchine di Turing e non di stringhe.

Chapter 5

Teoria della Complessità

Domanda: Quali problemi possiamo risolvere in pratica?
Quelli che ammettono algoritmi aventi tempo polinomiale

Supponiamo di lavorare per la posta e ci viene chiesto di risolvere il problema dei postini minimizzando il costo.
Non riusciamo a trovare nulla di che..

Potremmo andare dal capo a dire non ci riesco...

LICENZIATO!

Possiamo dire che non ci siamo riusciti, ma neanche molti altri scienziati prima di noi ci sono riusciti.

Quindi non si sa se vi è soluzione

5.1 Complessità algoritmi

INDECIDIBILE: nessun algoritmo possibile

CLASSE P: $O(n^k)$ possibile

NP-COMPLETEZZA: $O(n^k)$ non probabile

5.2 Riduzioni polinomiali

Supponiamo che possiamo risolvere X in tempo polinomiale. Quali altri problemi si possono risolvere in tempo polinomiale?

Il problema X si riduce polinomialmente al problema Y se arbitrarie istanze di X possono essere risolte usando:

- un numero polinomiale di passi di computazione più
- un numero polinomiale di chiamate ad un oracolo che risolve Y

$$X \leq_p Y$$

- Se Y ammette soluzione in tempo polinomiale allora anche X ammette soluzione in tempo polinomiale
- Se X non ammette soluzione in tempo polinomiale anche Y non l'ammetterebbe

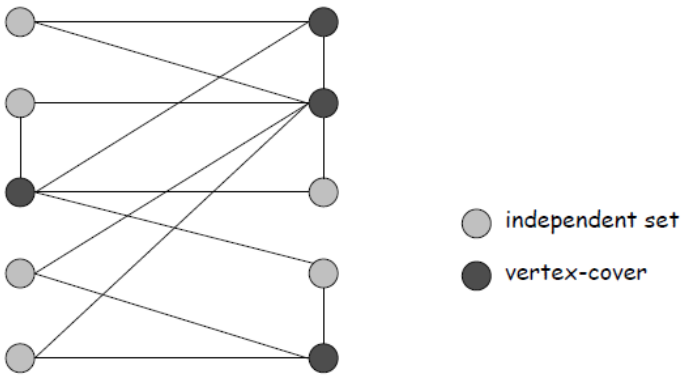
Equivalenza
Se $X \leq_p Y$ e $Y \leq_p X$ e $X \equiv Y$

5.2.1 Strategie di riduzione

- Riduzione mediante equivalenza semplice
- Riduzione da caso speciale a caso generale
- Riduzione mediante codifica con ”gadgets”

5.3 Riduzione a equivalenza semplice

Indipendent-Set: dato un grafo $G = (V, E)$ e un intero k, esiste un sottoinsieme di vertici $S \subseteq V$ tale che $|S| \geq k$, e per ogni arco al più nodo è in S?



$\exists$ un IS di dimensione ≤ 6? SI
$\exists$ un IS di dimensione ≤ 7? NO

Vertex-Cover: dato un grafo $G = (V, E)$ e un intero k, esiste un sottoinsieme di vertici $S \subseteq V$ tale che $|S| \leq k$, e per ogni arco, almeno uno dei suoi estremi è in S?

$\exists$ un VC di dimensione ≥ 4? SI
$\exists$ un VC di dimensione ≥ 3? NO

un Vertex-Cover sono i vertici che con i loro archi coprono tutto il grafo

Vertec-Cover e Indipendent-Set sono polinomialmente equivalenti

Vertex-Cover $\equiv_p$ Indipendent-Set

Dimostrazione: Mostriamo che S è un insieme indipendente sse $V - S$ è un Vertex-cover

$\Rightarrow$

- Sia S un qualsiasi indipendent-set
- consideriamo un arco generico (u,v)
- S insieme indipendente $\Rightarrow u \notin S$ o $v \notin S$
  $\Rightarrow u \in V - S$ o $v \in V - S$

Chapter 6

Bibliografia