

Fondamenti di Informatica

Università di Verona Imbriani Paolo - VR500437 Professor Isabella Mastroeni October 21, 2025

Contents

\mathbf{Cos}	sa è l'informatica?	3
1.1	Perché la calcolabilità	3
	1.1.1 Macchina di Turing	3
	1.1.2 Limiti dell'informatica	4
1.2	Basi di logica	4
1.3	Nozioni sugli insiemi	5
1.4	Nozioni sulle relazioni	5
1.5	Nozioni sulle funzioni	6
Fun	zioni calcolabili	6
2.1	Quale funzioni numerabili ci sono?	7
Pri	ncipio di induzione	7
3.1	Linguaggi formali	9
3.2	Funzioni e insiemi	11
Ling	guaggi Regolari	12
4.1	Automa a stati finiti determinsitici (DFA)	12
4.2	Come si dimostra che un linguaggio è regolare?	13
4.3	Automi a stati finiti non deterministici (NFA)	18
	4.3.1 Automi non deterministici con ε -transizioni (ε -NFA)	20
Esp	ressioni Regolari	21
5.1	Proprietà di LR (linguaggi regolari)	24
	5.1.1 Proprietà di chiusura	24
	5.1.2 Proprietà di decidibilità	24
	5.1.3 Esistenza dell'automa minimo	25
	5.1.4 Pumping Lemma	29
	1.1 1.2 1.3 1.4 1.5 Fun 2.1 Pri: 3.1 3.2 Lin 4.1 4.2 4.3	1.1.1 Macchina di Turing 1.1.2 Limiti dell'informatica 1.2 Basi di logica 1.3 Nozioni sugli insiemi 1.4 Nozioni sulle relazioni 1.5 Nozioni sulle funzioni 1.5 Nozioni sulle funzioni Funzioni calcolabili 2.1 Quale funzioni numerabili ci sono? Principio di induzione 3.1 Linguaggi formali 3.2 Funzioni e insiemi Linguaggi Regolari 4.1 Automa a stati finiti determinsitici (DFA) 4.2 Come si dimostra che un linguaggio è regolare? 4.3 Automi a stati finiti non deterministici (NFA) 4.3.1 Automi non deterministici (NFA) 4.3.1 Automi non deterministici con ε-transizioni (ε-NFA) Espressioni Regolari 5.1 Proprietà di LR (linguaggi regolari) 5.1.1 Proprietà di chiusura 5.1.2 Proprietà di decidibilità 5.1.3 Esistenza dell'automa minimo

1 Cosa è l'informatica?

La domanda chiave di questo corso è: "Cosa è l'informatica?". Ci sono diversi definizioni a seconda del contesto, ma in generale l'informatica è lo studio dei processi che trasformano l'informazione. Possiamo vedere, storicamente, diverse definizioni come quella in inglese come "Computer Science" che vede l'informatica come studio della calcolabilità, della computazione e dell'informazione.

1.1 Perché la calcolabilità

Si studia la calcolabilità perché ci aiuta a capire cosa possiamo fare con gli strumenti che abbiamo. Quando descriviamo un programma, quanto tempo ci mette e quanto spazio utilizza è una domanda importante per capire se il programma è efficiente o meno. Anche in senso dei linguaggi di programmazione per capire se usiamo quello giusto per il problema che stiamo cercando di risolvere. Chiaramente è un qualcosa che in continuo sviluppo perché si evolve in base alla tecnologia che abbiamo a disposizione.

Uno dei pionieri è stato **Hilbert** che si chiedeva se la matematica fosse formalizzabile come insieme finito (non contraddittorio) di assiomi? Godel dimostrò che non è possibile rappresentare la matematica come un insieme finito di assiomi in maniera non contradditoria, dicendo che in ogni sistema formale ci sono proposizioni vere che non possono essere dimostrate all'interno del sistema.

Nel tentativo di rispondere a queste (ed altre) domande si è costruito un modello che permette di comprendere profondamente il agionamento computazionale, permettendo di applicarlo ad ogni disciplina. Cosa è calcolabile e cosa non lo è?

1.1.1 Macchina di Turing

Anche Turing si pose questa domanda e propose la **Macchina di Turing** come modello di calcolo. Una sola macchina (programmabile) per tutti i problemi. La macchina è universale (interprete):

$$Init(P,x) = \begin{cases} P(x) & \text{se } P \text{ è un programma che termina} \\ \uparrow & \text{se } P \text{ è un programma che non termina} \end{cases}$$

Se un problema è intuitivamente calcolabile, allora esisterà una macchina di Turing (o un dispostivo equivalente, come il computer) in grado di risolverlo, cioè calcolarlo. I modelli equivalenti possono essere:

- Lambda calcolo
- Funzioni ricorsive

• Linguaggi di programmazione (Turing completi)

I problemi non calcolabili sono infinitamente più numerosi di quelli calcolabili.

1.1.2 Limiti dell'informatica

L'informatica ha più limiti di quanto si possa pensare, definiti dall'equivalenza di Turing e l'incompletezza di Godel che ci dicono che non possiamo risolvere tutti i problemi. Ci sono anche limiti fisici e tecnologici come:

- Dati non osservabili (teorema di Shannon)
- Dati non controllabili (velocità della luce)

1.2 Basi di logica

Alcune nozioni di logica che ci serviranno in seguito:

- Linguaggio del primo ordine:
 - Simboli relazionali (p, q, ...)
 - Simboli di funzione (f, g, ...)
 - Simboli di costante (c, d, ...)

• Simboli logici:

- Parentesi (,) e virgola
- Insieme numerabile di variabili (v, x, ...)
- Connettivi logici $(\neg, \land, \lor, \rightarrow, \leftrightarrow)$
- Quantificatori (\forall, \exists)

• Termini:

- Variabili
- Costanti
- -f simbolo di funzione m-ario t_1, t_2, \ldots, t_m termini, allora $f(t_1, t_2, \ldots, t_m)$ è un termine.
- Formula atomica: p simbolo di relazione n-ario, t_1, t_2, \ldots, t_n termini, allora $p(t_1, t_2, \ldots, t_n)$ è una formula atomica.

• Formula:

- Formula atomica
- $-\phi$ formula, allora $\neg \phi$ è una formula
- $-\phi \in \psi$ formule, allora $(\phi \land \psi)$, $(\phi \lor \psi)$, $(\phi \to \psi)$, $(\phi \leftrightarrow \psi)$ sono formule.
- $-\phi$ formula e v variabile, alloar e $\forall v.\phi$ e $\exists v.\phi$ sono formule.

1.3 Nozioni sugli insiemi

- $x \in A$ signfiica che x è un elemento dell'insieme A
- $\{x|P(x)\}$ si identifica insieme costuito dagli x che soddisfano la proprietà (o predicato) P(x)
- $A\subseteq B$ significa che A è un sottoinsieme di B se ogni elemento di A è anche in B
- $\mathcal{P}(S)$ denota l'insieme delle parti di S, ovvero l'insieme di tutti i sottoinsiemi di S ($\mathcal{P}(S) = \{X | X \subseteq S\}$)
- $A \setminus B = \{x | x \in A \land x \notin B\}, A \cup B = \{x | x \in A \lor x \in B\}, A \cap B = \{x | x \in A \land x \in B\}$
- |A| denota la cardinalità di A, ovvero il numero di elementi in A.
- \bar{A} denota il complemento di A, ovvero $x \in A \leftrightarrow x \notin A$

1.4 Nozioni sulle relazioni

- Prodotto cartesiano: $A_1 \times A_2 \times \cdots \times A_n = \{ \langle a_1, a_2, \dots, a_n \rangle | a_1 \in A_1, \dots, a_n \in A_n \}$
- Una **RELAZIONE** (binaria) è un sottoinsieme del prodotto cartesiano di (due) insiemi; dati A e B, $R \subseteq A \times B$ è una relazione su A e B
 - Riflessiva: $\forall a \in S \text{ si ha che } aRa$
 - Simmetrica: $\forall a, b \in S \text{ se } aRb \text{ allora } bRa$
 - Antisimmetrica: $\forall a, b \in S \text{ se } aRb \text{ e } bRa \text{ allora } a = b$
 - Transitiva: $\forall a,b,c \in S$ se aRb e bRc allora aRc
- Per ogni relazione $R \subseteq S \times S$ la chiusura transitiva di R è il più piccolo insieme R^* tale che $\langle a,b \rangle \in R \land \langle b,c \rangle \in R \rightarrow \langle a,c \rangle \in R^*$
- Una relazione è detta **totale** su S se $\forall a,b \in S$ si ha che $aRb \vee bRa$
- Una relazione R di di equivalenza è una relazione binaria riflessiva, simmetrica e transitiva.
- Una relazione binaria $R \subseteq S \times S$ è un **pre-ordine** se è riflessiva e transitiva.
- ${\cal R}$ è un ordine parziale se è un pre-ordine antisimmetrico.
- $x \in S$ è **minimale** rispetto a R se $\forall y \in S.y \ Rx$ (ovvero $\neg(yRx)$)
- $x \in S$ è **minimo** rispetto a R se $\forall y \in S.xRy$
- $x \in S$ è massimale rispetto a R se $\forall y \in S.x \ Ry$ (ovvero $\neg(xRy)$)
- $x \in S$ è massimo rispetto a R se $\forall y \in S.yRx$

1.5 Nozioni sulle funzioni

- Una relazione f è una funzione se $\forall a \in A$ esiste uno ed un solo $b \in B$ tale che $(a,b) \in f$
- A dominio e B codominio di f. Il range di f è l'insieme di tutti i valori che f può assumere.
- f è iniettiva se $\forall a_1, a_2 \in A$ se $a_1 \neq a_2$ allora $f(a_1) \neq f(a_2)$
- Se $f:A\mapsto B$ è sia iniettiva che suriettiva allora è biiettiva e quindi esiste $f^{-1}:B\mapsto A$

2 Funzioni calcolabili

Un insieme è a tutti gli effetti una proprietà che dato un oggetto stabilisce se esso all'interno di insieme o no.

Problemi
$$\equiv$$
 Funzioni $f: \mathbb{N} \to \mathbb{N}$

Ci chiediamo se questa funzioni siano tutte calcolabili (intuitivamente). Da il teorema che abbiamo citato nei precedenti paragrafi (quello dell'incompletezza) sappiamo che non è così. Cerchiamo di vedere insiemisticamente perché questo è giustificato.

☼ Definizione 2.1: Intuitivamente Calcolabile

Qualcosa che è **intuitivamente calcolabile** è qualcosa è che riusciamo a descrivere attraverso un algoritmo, ovvero una sequenza finita di passi discreti elementari.

La funzione di tipo:

$$f: \mathbb{N} \to \mathbb{N}$$

è un **insieme** di associazioni input-output.

Esempio 2.1

$$f = \text{quadrato} = \{(0,0), (1,1), (2,4), (3,9), (4,16), \dots\}$$
$$= \{(x, x^2) \mid n \in \mathbb{N}\}$$

Quindi f è un insieme di coppie in $\mathbb{N} \times \mathbb{N}$. Quindi $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$ (cardinalità di $\mathbb{N} \times \mathbb{N}$ Quindi

$$f \subseteq \mathbb{P}(\mathbb{N} \times \mathbb{N}) = \mathbb{P}(\mathbb{N})$$

Per esempio se:

$$A = \{1, 2, 3\}$$
 allora $\mathbb{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}\}$

dove $|\mathbb{P}(A)| = 2^{|A|}$.

$$|\mathbb{N}| = \omega < |P(\mathbb{N})|) = |\mathbb{R}|$$

e quindi l'insieme delle funzioni **non è numerabile**.

2.1 Quale funzioni numerabili ci sono?

 Σ =alfabeto finito di simboli che uso per il programma/algoritmo

$$\Sigma = s_1, s_2, s_3, \dots$$

quindi un programma non è nient'altro che un sottoinsieme finito di Σ^* (tutte le stringhe finite che posso formare con l'alfabeto).

$$\Sigma = a, b, c$$

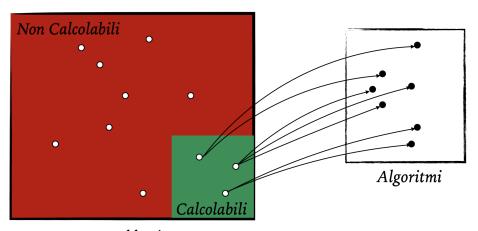
$$\Sigma^* = \{\emptyset, a, b, c, ab, ba, ac, cd, bc, cb, \dots\}$$

in questo caso, la sequenza di simboli in Σ^* è numerabile, perché

$$|\Sigma^*| = |\mathbb{N}|$$

|Programmi in
$$\mathbb{N}$$
| = $|\Sigma^*|$ = $|\mathbb{N}|$

e di conseguenza l'insieme dei programmi è numerabile. Una veloce constatazione che possiamo fare è vedere quindi che l'insieme delle funzioni calcolabili è numerabile, perché ogni funzione calcolabile è associata ad almeno un programma che la calcola.



Problemi

3 Principio di induzione

Il principio di induzione ha senso solo su insieme infiniti e serve per dimostrare che una proprietà vale per tutti gli elementi. Esistono due metodi di induzione:

- Induzione matematica
- Induzione strutturale

Tratteremo nello specifico caso in questo corso l'induzione matematica. Partiamo da un insieme A infinito con una relazione <:(A,<) è una relazione d'ordine non riflessiva perché non è minore stretto. Utilizziamo l'induzione matematica e quindi $A=\mathbb{N}$ e < è l'ordinamento stretto tra numeri. Una relazione d'ordine deve essere ben fondata vuol dire che non esistono catene discendenti infinite. Una catena discendente è una sequenza infinita di elementi

$$a_0 > a_1 > a_2 > a_3 > \dots$$

In un tipo di relazione riflessiva è sicuramente non ben fondata perché posso fare continuare ad inserire lo stesso numero all'infinito.

m minimale in $A: b \in A$ è minimale se $\forall b' < b.b' \notin A$

ℰ Esempio 3.1

Se prendiamo $\{1,2,3\}$ con relazione d'ordine di contenimento allora esistono più minimali come $\{1,2\}$ o $\{2,3\}$. Quindi se $A=\mathbb{N} \Longrightarrow \exists b$ minimo $\forall x\subseteq \mathbb{N}$.

Quindi preso A insieme ben fondato con ordinamento < allora: π proprietà definita sugli elementi di

$$A: \pi \subseteq A \text{ allora } \forall a \in A.\pi(a) \iff \forall a \in A.[[\forall b < a.\pi(b)] \to \pi(a)]$$

Se dimostriamo π per ogni elemento più piccolo di a allora π vale per a.

$$Base_A = \{a \in A | a \text{ minimale}\}\$$

Quindi

$$\overbrace{\forall A \in Base_A \ . \ \pi(a)}^{\text{Base}} \ \underbrace{\forall a \in A.Base_A}_{\text{passo induttivo}}. \ \underbrace{\forall b < a.\pi(b)}_{\text{tesi}} \to \underbrace{\pi(a)}_{\text{tesi}}$$

Esempio 3.2

Prendiamo come esempio il seguente enunciato:

$$\forall n \in \mathbb{N}, \ \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Base:
$$n=1$$

$$A = \mathbb{N} \setminus \{0\}$$
 $Base_A = \{1\} \to \sum_{i=1}^{1} i = 1$

$$\sum_{i=1}^{1} i = 1$$

$$= n(n+1)/2$$

$$= \frac{1(1+1)}{2} = 1$$

Caso base dimostrato.

Passo induttivo: prendo $n \in \mathbb{N}$ e applico l'ipotesi induttiva: per ogni k < n vale la tesi.

Tesi da dimostrare:
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n$$

Sappiamo sicuramente che n-1 < n e per ipotesi induttiva:

$$\sum_{i=1}^{n-1} i + n = \frac{(n-1)(n-1+1)}{2} + n = \frac{n(n-1)}{2} + n =$$

$$\frac{n(n-1)+2n}{2} = \frac{n(n-1+2)}{2} = \frac{n(n+1)}{2} \quad \Box$$

e quindi la tesi vale perché siamo arrivati alla stessa espressione che volevamo dimostrare.

3.1 Linguaggi formali

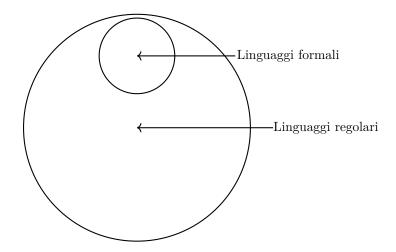
♦ Definizione 3.1: Linguaggio formale

Un linguaggio formale è un insieme di stringhe composte da simboli in un alfabeto finito Σ .

 Σ^* denota il linguaggio di tutte le stringhe dell'alfabeto Σ , se Σ non è vuota allora Σ^* è infinito e numerabile. Solitamente un linguaggio formale \mathcal{L} è un sottoinsieme di Σ^* tipicamente infiniti ma non è necessario:

$$\mathcal{L}\subseteq \Sigma^*$$

I linguaggi finiti sono sicuramente regolari perché posso sempre costruire un automa a stati finiti che li riconosce.



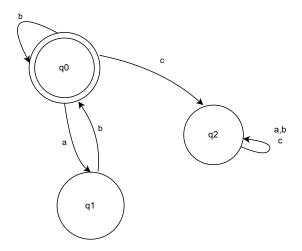


Figure 1: Esempio di automa a stati finiti

Come abbiamo detto in precedenza, una funzione è *calcolabile* se possiamo pensare ad un algoritmo per calcolarla. Tra le funzioni calcolabili, ci sono quelle totali, ovvero quelle che terminano per ogni input.

Esempio 3.3

Pensiamo ad una funzione $f \subseteq \mathbb{N} \times \mathbb{N}$, la possiamo descrivere come associazione di coppie:

$$f(0) = 1, f(1) = 1, f(2) = 2, f(3) = 3, f(4) = 5, f(5) = 8, \dots$$

Tuttavia servirebbe scrivere una quantità infinita di coppie per descrivere la funzione.

Quindi proviamo a farlo ma stavolta ricorsivamente.

$$\begin{cases} f(0) = 1 = f(1) \\ f(x+2) = f(x+1) + f(x) \end{cases}$$

3.2 Funzioni e insiemi

In realtà ci sta un forte legame tra funzioni e insiemi perché per esempio dovessimo prendere una funzione $f: \mathbb{N} \to \mathbb{N}$, questa funzione può essere vista come un linguaggio $L_f = \{1^{f(x)} | x \in \mathbb{N}\}$ che è l'insieme delle stringhe composte da f(x) simboli 1. Il nostro linguaggio $\Sigma = \{1\}$ e quindi il linguaggio è un sottoinsieme di Σ^* . Il nostro linguaggio ci può dire se un oggetto in input fa parte o no dell'insieme. Parliamo di **insiemi** invece che di funzioni dove gli elementi dell'insieme dipendono dal calcolo della funzione.

Esempio 3.4

Se io dovessi scrivere una funzione costante del tipo f(x)=2. In questo caso riconosciamo che il linguaggio L_f è finito perché

$$L_f = \{1\}$$

L'unica combinazione possibile è una singola stringa. Il linguaggio si dice finito che è sottoinsieme di linguaggi totali.

Esempio 3.5

In questo caso la funzione f(x) = 2x è una funzione lineare. Ho bisogno di una memoria finita. Mi sposto tra queste informazioni per determinare se la stringa in input fa parte o no del linguaggio. Questa funzione è anche detta **regolare**.

Esempio 3.6

 $f(\sigma) = \sigma_{\text{reverse}}$ quindi per esempio:

$$f(abc) = cba$$

Tuttavia per questa funzione non mi basta più una memoria finita ma illimitata essendo che non posso sapere a priori quanta memoria abbia bisogno la funzione (è sufficiente uno stack). Questo tipo di funzione è detta anche **context free**.

Esempio 3.7

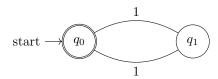
Nel caso di $f(x) = x^2$ sappiamo sicuramente che possiede un output preciso ma avrei bisogno di una memoria illimitata per rappresentarla.

4 Linguaggi Regolari

4.1 Automa a stati finiti determinsitici (DFA)

Quando parliamo di memoria è facilmente codificabile in termini di stati. Dal punto di vista grafico possiamo rappresentarli come nodi collegati da archi. Prendiamo per esempio il linguaggio L_f di prima:

$$L_f = \{1^{2n} \mid n \in \mathbb{N}\}$$
$$\Sigma = 1$$



🖒 Definizione 4.1: Automa a stati finiti

$$M = (Q, \Sigma, \delta, q_0, F)$$

è un automa a stati finiti deterministico dove:

- Q è un insieme finito di stati (ogni stato rappresenta "un informazione")
- Σ è un alfabeto finito di simboli
- $\delta: Q \times \Sigma \to Q$ è la funzione di transizione (descrive come evolve il calcolo a partire dallo stato raggiunto e dal simbolo letto)
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme di stati finali (o di accettazione)

$\Sigma \backslash Q$	q_0	q_1
1	q_1	q_0

Ci sta poi la funzione $\hat{\delta}:Q\times\Sigma^*\to Q$ che estende la funzione di transizione e descrive lo stato

che raggiungiamo leggendo una sequenza di simboli.

$$\begin{cases} \hat{\delta}(q,\epsilon) = q \\ \hat{\delta}(q,wa) = \hat{\delta}(\delta(q,w),a) \end{cases} \quad w \in \Sigma^*, a \in \Sigma$$

anche chiamata chiusura transitiva di δ . Quindi mi permette di calcolare lo stato raggiunto leggendo una stringa di simboli.

4.2 Come si dimostra che un linguaggio è regolare?

Esempio 4.1

Prendiamo come esempio il linguaggio:

$$L = \{ \sigma \mid \sigma \text{ contiene almeno due } 1 \}$$

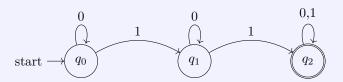
$$\Sigma = \{0, 1\}$$

Per esempio:

$$110, 101, 111, 011, 1001 \in L$$

$$0,00,000,01,10,0000 \notin L$$

Quindi intuitivamente come possiamo definire l'automa?



Un linguaggio L è riconosciuto da M (Automa a stati finiti deterministico o DFA) se L = L(M) dove L(M) è il linguaggio di n definito come:

$$L(M) = \{ \sigma \in \Sigma^* \mid \hat{\delta}(q_0, \sigma) \in F \}$$

Sono tutte le stringhe che partendo da q_0 (stato iniziale) raggiunge uno stato finale.

♦ Definizione 4.2

Per dimostrare che L è regolare dobbiamo costruire M (almeno un M) e dimostrare che L(M) = L.

$$L = L(M) \equiv L \subseteq L(M) \in L(M) \subseteq L$$

1.

$$L \subseteq L(M) \equiv \sigma \in L \to \sigma \in L(M)$$

$$\sigma \in L \to \hat{\delta}(q_0, \sigma) \in F$$

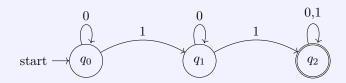
2.

$$\begin{split} L(M) \subseteq L &\equiv \sigma \in L(M) \to \sigma \in L \\ &\hat{\delta}(q_0,\sigma) \in F \to \sigma \in L \equiv \\ &\sigma \notin L \to \hat{\delta}(q_0,\sigma) \notin F \end{split}$$

Dimostriamo per induzione sulla lunghezza delle stringhe $\sigma \in \Sigma^*$ che se $\sigma \in L$ allora $\hat{\delta}(q_0, \sigma) \in F$ e $\sigma \notin L$ allora $\hat{\delta}(q_0, \sigma) \notin F$.

Esempio 4.2

 $L = \{ \sigma \mid \sigma \text{ contiene almeno due } 1 \}$



Dimostriamo per induzione sulla lunghezza delle stringhe $\sigma \in \Sigma^*$ che se $x \in L$ allora $\hat{\delta}(q_0, x) \in F$ e se $x \notin L$ allora $\hat{\delta}(q_0, x) \notin F$.

 $|\sigma|=0$ non è **mai** sufficiente come base, ma è eventualmente la base **solo** per una delle due dimostrazioni. Bisogna quindi prendere la lunghezza più piccola che permette di avere sia $\sigma \in L$ che $\sigma \notin L$, in questo caso è $|\sigma|=2$. Per ogni σ tale che $|\sigma|<2$ $\sigma \notin L$ perchè non può contenere due 1 e non è riconosciuta da M dove il primo stato finale è raggiunto leggendo almeno due simboli.

$$\varepsilon \in L \quad \varepsilon \notin L$$

• Base: Controlliamo ogni stringa di lungheezza minima nel linguaggio per provare il caso base

$$\begin{cases} \sigma &= 11 \in L \text{ e } \hat{\delta}(q_0, 11) = q_2 \in F \\ \sigma &= 10 \notin L \text{ e } \hat{\delta}(q_0, 10) = q_1 \notin F \\ \sigma &= 01 \notin L \text{ e } \hat{\delta}(q_0, 01) = q_1 \notin F \\ \sigma &= 00 \notin L \text{ e } \hat{\delta}(q_0, 00) = q_0 \notin F \end{cases}$$

• Passo induttivo: Dimostriamo l'ipotesi induttiva, cioè la tesi con un limite

fissato:

$$\forall \sigma \in \Sigma^* : |\sigma| \le n : \begin{cases} \sigma \in L & \Rightarrow \hat{\delta}(q_0, \sigma) \in F \\ \sigma \notin L & \Rightarrow \hat{\delta}(q_0, \sigma) \notin F \end{cases}$$

Vogliamo dimostrare se $|\sigma| = n + 1$ (la successiva stringa che posso considerare) allora vale $\begin{cases} \sigma \in L \Rightarrow \hat{\delta}(q_0, \sigma) \in F \\ \sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) \notin F \end{cases}$

Dimostrazione:

$$|\sigma| = n + 1 \rightarrow \sigma = \sigma' 1 \lor \sigma = \sigma' 0$$

– Supponiamo che σ appartiene al linguaggio e termini con 1:

$$\sigma \in L \wedge \sigma = \sigma' 1 \rightarrow$$

* Se $\sigma' \in L$ applico l'ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_2$$

$$\hat{\delta}(q_0, \sigma) \stackrel{\sigma = \sigma' 1}{=} \hat{\delta}(q_0, \sigma' 1)$$

$$= \delta(\hat{\delta}(q_0, \sigma'), 1)$$

$$=\delta(q_2,1)=q_2\in F$$

* Se $\sigma' \not\in L$ allora σ' contiene esattamente un 1:

$$\hat{\delta}(q_0, \sigma') = q_1$$

$$\hat{\delta}(q_0, \sigma' 1) = \delta(q_1, 1) = q_2$$

- Supponiamo che σ appartiene al linguaggio e termini con 0:

$$\sigma \in L \wedge \sigma = \sigma' 0$$

Per definizione di L abbiamo che

$$\sigma \in L \land \sigma = \sigma' 0 \Rightarrow \sigma' \in L$$

Dimostriamo l'ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_2$$

allora

$$\hat{\delta}(q_0, \sigma) = \hat{\delta}(q_0, \sigma'0)$$

$$= \delta(\hat{\delta}(q_0, \sigma'), 0)$$

$$= \delta(q_2, 0) = q_2 \in F$$

– Supponiamo che σ non appartiene al linguaggio e termini con 1:

$$\sigma \not\in L \wedge \sigma = \sigma' 1$$
 (ha esattamente un 1)

 $\downarrow \downarrow$

 $\sigma \not\in L$ non ha 1

- Supponiamo che σ non appartiene al linguaggio e termini con 0:

$$\sigma \not\in L \wedge \sigma = \sigma' 0$$

1

 $\sigma' \not\in L$

ℰ Esempio 4.3 (Esercizio da esame)

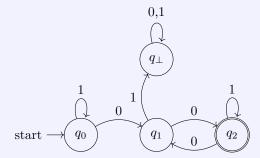
$$L = \{ \sigma \in \Sigma^* \mid \exists n \ge 1 : \sigma = 0^n \to n \ge 2 \}$$

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

Ovvero ogni sequenza di 0 nella stringa deve essere di lunghezza pari. Per esempio:

$$101 \not\in L, 1111 \in L$$

 $1001 \in L,000 \not\in L,0000 \in L,010101 \not\in L$



- q_0 : non abbiamo letto 0
- q_1 : rappresenta una sequenza di 0 consecutivi di lunghezza dispari
- q_2 : sequenza di 0 di lunghezza pari.

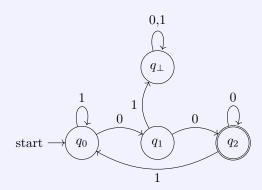
Tesi: $\begin{cases} \sigma \in L \text{ e non contiene } 0 & \to \hat{\delta}(q_0, \sigma) = q_0 \\ \sigma \in L \text{ e contiene } 0 & \to \hat{\delta}(q_0, \sigma) = q_2 \\ \sigma \notin L \text{ e contiene una sequenza finali di } 0 & \to \hat{\delta}(q_0, \sigma) = q_1 \\ \sigma \notin L \text{ e contiene una sequenza dispari di } 0 \text{ seguiti da un } 1 & \to \hat{\delta}(q_0, \sigma) = q_\perp \end{cases}$

Esempio 4.4

$$L = \{ \sigma \in \Sigma^* \mid \exists n \geq 1 \; . \; \sigma = 0^n \to n \geq 2 \}$$

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

Se σ non contiene 0 allora $\sigma \in L$.



- q_0 : non contiene 0 oppure tutte le sequenze di 0 sono lunghe almeno 2.
- q_1 : esattamente uno 0.
- q_2 : almeno due 0.

Tesi: $\begin{cases} \sigma \in L \text{ e } \sigma = \sigma' 1 \to \hat{\delta}(q_0, \sigma) = q_0 \\ \sigma \in L \text{ e } \sigma = \sigma' 0 \to \hat{\delta}(q_0, \sigma) = q_2 \\ \sigma \notin L \text{ e } \sigma = \sigma' 0 \text{ dove l'ultima seq di 0 è esattamente lunga } 1 \to \hat{\delta}(q_0, \sigma) = q_1 \\ \sigma \notin L \text{ e } \sigma \text{ contiene una sequenza lunga 1 di } 0 \to \hat{\delta}(q_0, \sigma) = q_\perp \end{cases}$

4.3 Automi a stati finiti non deterministici (NFA)

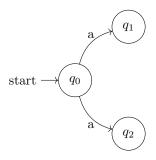
♦ Definizione 4.3

Un automa a stati finiti non deterministico è una 5-upla:

$$N = (Q, \Sigma, \delta, q_0, F)$$

dove:

- Q è un insieme finito di stati
- Σ è un alfabeto finito di simboli
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme di stati finali (o di accettazione)
- $\delta: Q \times \Sigma \to \mathcal{P}(Q)$ è la funzione di transizione dove per ogni coppia di stato-simbolo ho un insieme di stati potenzialmente raggiungibili.



$$\delta(q_0, a) = \{q_1, q_2\} \subseteq Q$$
$$\emptyset \in \mathcal{P}(Q)$$

- È possibile che non esistano coppie associate al vuoto
- Non è obbligatorio avere un arco uscente per ogni simbolo di Σ .

Definiamo $\hat{\delta}: \hat{\delta}: Q \times \Sigma^* \to \mathcal{P}(Q)$

$$\begin{cases} \hat{\delta}(q,\epsilon) = \{q\} \\ \hat{\delta}(q,wa) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p,a) \end{cases}$$

Linguaggio riconosciuto:

$$L(N) = \{ \sigma \in \Sigma^* \mid \hat{\delta}(q_0, \sigma) \cap F \neq \emptyset \}$$

Teorema 4.3.1 Teorema di Rabin-Scott

Se abbiamo $N=(Q,\Sigma,\delta,q_0,F)$ un NFA allora esiste M DFA tale che L(N)=L(M).

Quindi se M DFA allora N è un particolare NFA.

Proof. Preso $N=(Q,\Sigma,\delta,q_0,F)$ costruiamo $M=(Q',\Sigma,\delta',q_0',F')$ dove:

$$Q' = \mathcal{P}(Q)$$

$$Q = \{q_0, q_1, q_2\}$$

$$Q' = \{\varnothing, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}\}$$

$$q'_0 = \{q_0\}$$

$$F = \{q_2\}$$

$$F' = \{\underbrace{P \subseteq Q}_{P \in \mathcal{P}(Q)} \mid P \cap F \neq \varnothing\}$$

$$F' = \{\{q_2\}, \{q_1q_2\}, \{q_0q_2\}, \{q_0q_1q_2\}\}\}$$

$$\delta'(P, a) = \bigcup_{q \in P} \delta(q, a) \in \mathcal{P}(Q)$$

$$\delta'(q'_5, 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$$

$$= \{q_0, q_2\} \cup \{q_0, q_1, q_2\}$$

$$= \{q_0, q_1, q_2\}$$

Dimostriamo:

1.
$$\hat{\delta}'q_0', \sigma = \hat{\delta}(q_0, \sigma)$$

2.
$$\sigma \in L(N)$$
 sse $\sigma \in L(M)$

1. Per induzione su $|\sigma|$:

Se $\sigma = \epsilon$ allora: $\hat{\delta}'(q_0', \epsilon) = q_0' = \{q_0\} = \hat{\delta}(q_0, \epsilon)$ per le definizioni.

Se $\sigma=\sigma'a$: $\hat{\delta}'(q_0',\sigma'a)=\hat{\delta}(\hat{\delta}'(q_0,\sigma'),a)$. Per ipotesi induttiva:

$$\hat{\delta}'(q_0', \sigma') = \hat{\delta}(q_0, \sigma')$$

2.
$$\sigma \in L(N)$$
 sse $\sigma \in L(M)$

$$\sigma \in L(N) \Leftrightarrow \hat{\delta}(q_0, \sigma) \cap F \neq \varnothing$$

$$\Leftrightarrow \hat{\delta}'(q'_0, \sigma) \cap F \neq \varnothing$$

$$\Leftrightarrow \hat{\delta}'(q'_0, \sigma) \in F'$$

$$\Leftrightarrow \sigma \in L(M) \text{ def. linguaggio accettato in DFA} \quad \square$$

Una volta aver raggiunto un risultato di automa a stati finiti deterministico non è detto che sia il migliore che possiamo trovare. Possiamo eliminare gli stati non raggiungibili e ottenere un automa equivalente più piccolo (**DFA minimo**).

4.3.1 Automi non deterministici con ε -transizioni (ε -NFA)

Gli ε -NFA sono una variante degli NFA in cui possiamo cambiare stato senza leggere simboli.

$$q_1 \stackrel{\varepsilon}{\to} q_2$$

♣ Definizione 4.4

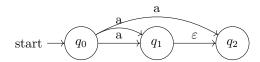
Un $\varepsilon\textsc{-NFA}$ è una 5-upla:

$$N = (Q, \Sigma, \delta, q_0, F)$$

dove:

- Q è un insieme finito di stati
- Σ è un alfabeto finito di simboli
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme di stati finali (o di accettazione)
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ è la funzione di transizione dove per ogni coppia di stato-simbolo ho un insieme di stati potenzialmente raggiungibili. La ε mi dice che possiamo cambiare stato senza leggere simboli.

Per esempio:



Definiamo ora:

$$\hat{\delta}: Q \times \Sigma^* \to \mathcal{P}(Q)$$

$$\varepsilon$$
 – closure : $Q \to \mathcal{P}(Q)$

La ε -closure(q) di uno stato q è l'insieme di stati raggiungibili da q seguendo archi etichettati con ε .

$$\varepsilon$$
 – closure : $\mathcal{P}(Q) \to \mathcal{P}(Q)$

$$\begin{cases} \hat{\delta}(q,\epsilon) = \varepsilon\text{-closure}(q) \\ \hat{\delta}(q,wa) = \varepsilon\text{-closure}\left(\bigcup_{p \in \hat{\delta}(q,w)} \delta(p,a)\right) \end{cases}$$

Esempio 4.5

La ε -closure di q' nell'automa sopra è:

$$\varepsilon$$
-closure $(q') = \{q', q''\}$

Il riconoscimento di un linguaggio è analogo a quello di un NFA:

$$L(N) = \{ \sigma \in \Sigma^* \mid \hat{\delta}(q_0, \sigma) \cap F \neq \emptyset \}$$

Teorema 4.3.2

Sia $N=(Q,\Sigma,\delta,q_0,F)$ un ε -NFA. Allora esiste un NFA N' tale che L(N)=L(N').

Ciò vuol dire che l'insieme dei linguaggi riconosciuti da ε -NFA coincide con quello degli NFA, che a sua volta coincide con i linguaggi regolari. Prendiamo $N = (Q, \Sigma, \delta, q_0, F)$ un ε -NFA. Costruiamo NFA $N' = (Q', \Sigma', \delta, q'_0, F')$ dove $\delta'(q, a) = \hat{\delta}(q, a)$ e $F' = \begin{cases} F \cup \{q_0\} & \text{se } \varepsilon\text{-closure}(q_0) \cap F \neq \varnothing \\ F & \text{altrimenti} \end{cases}$.

5 Espressioni Regolari

Le espressioni regolari sono un modo compatto per rappresentare i linguaggi. Le opreazioni che si possono fare sono:

- Unione: Siano $L_1, L_2 \subseteq \Sigma^*$ linguaggi : $L_1 \cup L_2 = \{ \sigma \mid \sigma \in L_1 \text{ oppure } \sigma \in L_2 \}$
- Concatenazione: Siano $L_1, L_2 \subseteq \Sigma^*$ linguaggi : $L_1 \cdot L_2 = \{\sigma_1 \sigma_2 \mid \sigma_1 \in L_1, \sigma_2 \in L_2\}$ che permette la concatenazione di tutte le stringhe di L_1 con tutte le stringhe di L_2 .
- Stella di Kleene: Sia $L \subseteq \Sigma^*$ un linguaggio : $L^* = \bigcup_{n \in \mathbb{N}} L^n$ dove L^n è la concatenazione di L con sè stesso n volte.

$$\begin{cases} L^0 = \{\varepsilon\} \\ L^{n+1} = L^n \cdot L \end{cases}$$

♦ Definizione 5.1

 Σ alfabeto, definiamo per induzione:

· Caso base:

- $-\ \varnothing \subseteq \Sigma^*$ è espressione regolare che rappresenta il linguaggio vuoto.
- $-\varepsilon$ è espressione regolare che rappresenta il linguaggio $\{\varepsilon\}\subseteq\Sigma^*$.
- $-a \in \Sigma$ è espressione regolare che rappresenta il linguaggio $\{a\} \subseteq \Sigma^*$.

• Passo induttivo:

- r,ssono espressioni regolari che rappresentono il linguaggio $R\subseteq \Sigma^{\alpha}$ e $S\subseteq \Sigma^*$ rispettivamente.
 - * (r) + (s) è espressione regolare che rappresenta il linguaggio $R \cup S$.
 - * (r)(s) è espressione regolare che rappresenta il linguaggio $R \cdot S$.
 - * $(r)^*$ è espressione regolare che rappresenta il linguaggio R^* .

Esempio 5.1

Prendiamo come esempio la seguente espressione:

$$1^* + 0^* + (10)^*$$

equivale:

$$\{1^n \mid n \in \mathbb{N}\} \cup \{0^n \mid n \in \mathbb{N}\} \cup \{(10)^n \mid n \in \mathbb{N}\}$$

Un'altra equivalenza interessante è:

$$L = 000, 111 \rightarrow (000 + 111)$$

$$L^* = (000 + 111)^*$$

Teorema 5.0.1 Teorema di equivalenza

Dato M DFA $(Q, \Sigma, \delta, q_0, F)$ allora esiste r espressione regolare t.c L(N) = L(r).

L regolare
$$\stackrel{def}{\Leftrightarrow} \exists \underbrace{M\ DFA}_{L(n)=L} \stackrel{thm}{\to} \exists r \in ER \mid L(r) = L$$

Teorema 5.0.2

Data r espressione regolare (ER) allora esiste un ε -NFA M tale che L(r) = L(N).

- $\varepsilon \leadsto q_0, L(N) = \{\varepsilon\}$ $\varnothing \leadsto q_0, q_1, L(N) = \varnothing$ $a \leadsto q_0 \to q_1, L(N) = \{a\}$
- r, s espressioni regolari per ipotesi induttiva $\exists N_1.L(n_1) = L(r)$ e $\exists N_2.L(n_2) = L(s)$.

$$r \: ER \Rightarrow M \: \varepsilon\text{-closure} \Longleftrightarrow M' \: \mathrm{DFA} \Longleftrightarrow L(M')$$
è regolare

Esempio 5.2

$$L = \{ \sigma \in \Sigma^* \mid \sigma \text{ contiene almeno due } 1 \}$$

per dimostrare che è regolare si costruisce il DFA M e si dimostra ceh L = L(M).

$$r = 0*10*10*$$

dove r non dimostra che L è regolare, perché dovremmo dimostrare L=L(r). L(r)è sicuramente regolare ma dobbiamo comunque dimostrare l'uguaglianza insiemistica L = L(r).

5.1 Proprietà di LR (linguaggi regolari)

5.1.1 Proprietà di chiusura

Rispetto a quali operazioni due linguaggi regolari sono chiusi? Questa proprietà è utile perché a volte mi serve studiare un linguaggio complesso come intersezioni di linguaggi più semplici. Operazioni: $*, \cup, \cdot, \cap, \hat{}$.

Teorema 5.1.1

I linguaggi sono chiusi rispetto a stella di Kleene, unione (finita) e concatenazione.

Quindi

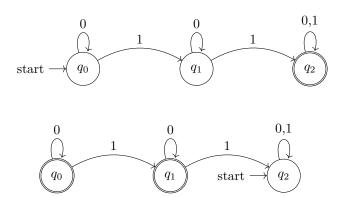
$$L_1, L_2 \operatorname{reg}$$

 L^* è reg. $L_1 \cup L_2$ reg L_1L_2 è regolare

$$\underbrace{a^3}_{\text{reg. finito}} \cdot \{b^n c^m \mid n, m \in \mathbb{N}\}$$

Teorema 5.1.2

I linguaggi regolari sono chiusi rispetto alla complementazione.



L'automa sopra disegnato riconosce il complemento.

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

per le leggi di Morgan, i linguaggi regolari sono chiusi per intersezione (finita).

5.1.2 Proprietà di decidibilità

L'insieme di stringhe accettate da un DFA (linguaggio regolare) DFA con M stati con n stati.

- 1. L(M) è $\neq \emptyset$ sse accetta almeno una stringa di lunghezza $\leq n$.
- 2. L(M) è infinito se accetta almeno una stringa lunga l con $n \leq l \leq 2n$.
- 3. $L(M_1) = L(M_2)$ cazzo piccolissimo
oo... 8======D

5.1.3 Esistenza dell'automa minimo

Fornisce strategie per costruire un automa minimo. Sia R relazione su $\Sigma: R \subseteq \Sigma \times \Sigma$. R è una relazione di equivalenza se:

• Riflessiva: $\forall a \in \Sigma.aRa$

• Simmetrica: $\forall a, b \in \Sigma . aRb \Rightarrow bRa$

• Transitiva: $\forall a, b, c \in \Sigma . aRb \land bRc \Rightarrow aRc$

Una relazione di equivalenza induce una partizione. $R \subseteq S \times S$ induce una partizione di S dove S_i sono una partizione di S se

$$S = S_1 \cup S_2 \cup \dots \cup S_n$$

e

- $\forall i, j \ S_i \cap S_j = \varnothing$
- $\forall a, b \in S_i.aRb$
- $\forall a \in S \, \forall b \in S_i. \neg (aRb)$

 S_i sono detti classi di equivalenza di R:

$$a \in R \Rightarrow [a]_R = \{b \mid aRb\}$$
classe di equivalenza di a

$$cRa \Rightarrow [a]_R \equiv [c]_R$$

Torniamo ora ai linguaggi regolari. Consideriamo un linguaggio $L \subseteq \Sigma^*$. Possiamo definire $R_L \subseteq \Sigma^* \times \Sigma^*$ come:

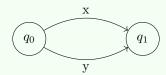
$$x, y \in \Sigma^*$$
 $xR_L y$ sse $\forall z \in \Sigma^* . xz$ sse $yz \in L$

Quello che diciamo quindi è che due stringhe sono in relazione se per ogni possibile estensione z la loro appartenenza a L è la stessa.

🖒 Definizione 5.2: Classe di equivalenza definita per gli automi

Consideriamo $M=(Q,\Sigma,\delta,q_0,F)$ un DFA. Possiamo definire $R_M\subseteq\Sigma^*\times\Sigma^*$ come:

$$x, y \in \Sigma^*.xR_m y \iff \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$$



Allora R_L e R_M sono relazioni di equivalenza.

🗘 Definizione 5.3: Relazione invariante destra

Una relazione di equivalenza R su Σ^* è invariante a destra se:

$$\forall x,y,z \in \Sigma^* \quad xRy \Rightarrow xzRyz$$

Quindi essere in relazione è invariante rispetto all'estensione della stringa verso destra. R_L e R_M sono relazioni invarianti destra.

☼ Definizione 5.4: Raffinamento

Siano R e S due relazioni di equivalenza su Σ^* . Diciamo che R raffina S se:

$$\forall x, y \in \Sigma^* \quad xRy \Rightarrow xSy$$

Quindi ogni classe di equivalenza di R è contenuta in una classe di equivalenza di S. Il numero di classi di equivalenza di R è maggiore del numero di classi di equivalenza di S.

☼ Definizione 5.5

Dato un DFA $M=(Q,\Sigma,\delta,q_0,F)$ ogni stato definisce un linguaggio.

$$q \in Q : L_q = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \}$$

R si dice di indice finito se il numero di classi di equivalenza indotte è finito.

Teorema 5.1.3 Teorema di Nyhill-Nerode

I seguenti enunciati sono equivalenti:

- 1. $L \subseteq \Sigma^*$ è regolare ovvero esiste un DFA M t.c. L(M) = L.
- 2. L è l'unione di classi di equivalenza indotte da una relazione di equivalenza R invariante destra con indice finito.
- 3. R_L è di indice finito.

Dimostriamo che:

$$(1) \stackrel{R_m}{\Longrightarrow} (2) \stackrel{R \text{ raffina } R_L}{\Longrightarrow} (3) \stackrel{\text{costruisce } M}{\Longrightarrow} (1)$$

Proof. $(1) \Rightarrow (2)$

Ipotesi: L linguaggio riconosciuto da $M = (Q, \Sigma, \delta, q_0, F)$ DFA.

$$L = L(M)$$

Tesi: $\exists R$ relazione di equivalenza invariante destra di indice finito t.c. L è l'unione di classi di equivalenza di R. Prendiamo $R = R_M - x R_m y$ sse $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$.

- 1. Il numero di classi di equivalenza di R_M è uguale al numero di stati |Q| ma per definizione Q è finito quindi R_M è di indice finito.
- 2. R_M è invariante destra:

$$\begin{split} L &= L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\} \\ &= \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \land q \in F\} \\ &= \bigcup_{q \in F} \{x \mid \hat{\delta}(q_0, x) = q\} \\ &= \bigcup_{q \in F} L_q \text{ dove } L_q \text{ è la classe di equivalenza di } R_M \end{split}$$

Quindi L è l'unione di classi di equivalenza di R_M .

Proof. $(2) \Rightarrow (3)$

Ipotesi: L è l'unione di classi di equivalenza indotte da una relazione di equivalenza R di indice finito.

Tesi: R_L è di indice finito. Possiamo dimostrare che R raffina R_L . Perché se fosse così allora il numero di classi di R sarebbe maggiore del numero di classi di R_L . Quindi se il numero di classi di R è finito allora sicuramente lo sarà anche R_L . Per dimostrare R raffinamento

di R_L dobbiamo dimostrare che $\forall x,y \in \Sigma^* \quad xRy \Rightarrow xR_Ly$. In maniera equivalente:

$$y \in [x]_R \Rightarrow y \in [x]_{R_L} \equiv [x]_R \subseteq [x]_{R_L}$$

Prendiamo xRy sapendo che R è invariante destra.

$$\forall z \in \Sigma^* \quad xRy \Rightarrow xzRyz$$

Quindi se L è l'unione di classi di equivalenza di R:

$$xRy \quad x \in L \text{ sse } y \in L$$

 $[x]_R\subseteq L$ oppure $[x]_R$ fuori da L

Quindi:

$$xRy \Longrightarrow x \in L \Longleftrightarrow y \in L$$

$$\Longrightarrow \forall z . xzRyz \Longrightarrow xz \in L \Longleftrightarrow yz \in L$$

$$\stackrel{\text{def } R_L}{\Longrightarrow} xR_L y$$

Queste dimostrazioni dato M generico, R_M è un raffinamento di R_L .

Proof. $(3) \Rightarrow (1)$

Ipotesi: R_L è di indice finito.

Tesi: L è riconosciuto da un DFA M.

Costruiamo M:

- $Q = \{[x]_{R_L} \mid x \in \Sigma^*\}$ è l'insieme delle classi di equivalenza di R_L (sono finite)
- $\Sigma =$ quello di L
- $q_0 = [\varepsilon]_{R_L}$
- $F = \{ [x]_{R_L} \mid x \in L \}$
- $\delta(q,a) = \delta([x]_{R_L},a) = [xa]_{R_L}$ questo è vero perché R_L è invariante destra. δ è una buona definizione perché indipendente dall'elemento che rappresenta la classe.

Ora dobbiamo dimostrare che L = L(M). Dimostrare per induzione che $\hat{\delta}([x], y) = [xy]$:

$$\hat{\delta}(q_0, x) = \hat{\delta}([\varepsilon]_{R_L}, x) = [x]_{R_L}$$

$$x \in L(M)$$
 sse $\hat{\delta}(q_0, x) \in F$
sse $\hat{\delta}([\varepsilon]_{R_L}, x) \in F$
sse $[x]_{R_L} \in F$
sse $x \in L$

Quindi L(M) = L.

Teorema 5.1.4

Dato L esiste M DFA t.c. L(M) = L e M ha il numero minimo di stati (quello costruito da R_L).

5.1.4 Pumping Lemma

Questo risultato è importante perché fornisce una condizione π necessaria alla regolarità.

$$L \text{ regolare} \Longrightarrow \pi$$

Questa forma si può riscrivere come:

$$\neg \pi \Longrightarrow L$$
non è regolare

Teorema 5.1.5 Pumping Lemma

Sia $L\subseteq \Sigma^*$ un linguaggio regolare. Allora esiste $k\in\mathbb{N}$ t.c. per ogni stringa $s\in L$ con $|s|\ge k$ può essere scritta come:

$$s = uvw$$

con:

- |v| > 0
- $|uv| \leq k$
- $\forall i \in \mathbb{N} . uv^i w \in L$