Palano Beatrice, ricevimento Giovedì (15^{30}/16^{30})

LINGUAGGI FORMALI E AUTOMI (LFA)

Esaminiamo il titolo del corso:

LINGUAGGIO

Intuitivamente possiamo definire un linguaggio come un insieme di frasi che permette la comunicazione tra più entità, in generale, con linguaggio s'intende la capacità d'uso e l'uso stesso di un qualunque sistema di simboli atti a comunicare. Per specificare un linguaggio abbiamo bisogno di due cose:

- Vocabolario: elenco di vocaboli appartenenti al linguaggio;
- Sintassi: regole con le quali si compongono i linguaggi, regole semantiche;

ESEMPIO DI LINGUAGGIO:

Linguaggio naturale (scritto o parlato): che consente di comunicare tra esseri umani, va osservata la profonda differenza tra il linguaggio parlato e il linguaggio scritto; nel primo caso, il messaggio è veicolato da un segnale acustico continuo, in cui è difficile rilevare la separazione tra lettere o parole; nel secondo caso il messaggio è codificato da una sequenza di caratteri tipografici e spazi bianchi. Un "testo", nel linguaggio scritto, può essere visto come una sequenza finita e prefissata di simboli scelti dall'insieme {a, …, z, A, …, Z, ., ;, !, ecc} ovviamente non ogni sequenza di simboli è riconosciuta quale "testo" del linguaggio. Un importante e difficile obiettivo della linguistica è di fornire una rigorosa descrizione di un linguaggio scritto, specificando quali frasi sono formate correttamente;

ESEMPIO DI LINGUAGGIO:

Linguaggio di programmazione: detti linguaggi artificiali, i linguaggi di programmazione sono importanti strumenti per la comunicazione tra l'uomo e la macchina. Ad esempio il linguaggio di programmazione C è descritto dalle sequenze di caratteri che un compilatore C accetta e riconosce per tale linguaggio;

ESEMPIO DI LINGUAGGIO:

Segnaletica stradale: è un linguaggio a simboli che comunica informazioni agli automobilisti;

FORMALE

Specifica rigorosa e precisa dei linguaggi mediante l'utilizzo di strumenti matematici, qual è il vantaggio di scrivere un linguaggio in maniera specifica? Il vantaggio è di poter trattare il linguaggio automaticamente, ossia trattare il linguaggio (tradurlo o interpretarlo) in modo meccanico, senza l'uomo;

ESEMPIO DI LINGUAGGIO FORMALE:

Il linguaggio di programmazione è un linguaggio formale, infatti, le specifiche con cui è scritto un programma sono rigorose e precise, consentendo un trattamento automatico da parte del compilatore (ossia della macchina meccanica che in assenza dell'uomo è in grado di trattare il linguaggio);

ESEMPIO DI LINGUAGGIO NON FORMALE:

Il linguaggio naturale è un linguaggio non formale, infatti, è difficile impostare una macchina (traduttore) che automaticamente riesca a tradurre da una lingua ad un'altra senza compromettere la giusta interpretazione del contesto.

AUTOMA

Macchina che consente di stabilire se una frase appartiene o meno ad un determinato linguaggio formale.
I linguaggi verranno formalizzati mediante:

- SISTEMI GENERATIVI: un esempio di sistema generativo è dato dalle grammatiche, per formalizzare un linguaggio vengono fornite delle regole (n'è un esempio un professore che, per insegnare la lingua inglese, fornisce specifiche e regole grammaticali da osservare per la corretta creazione di frasi);
- SISTEMA RICONOSCITIVO: un esempio di sistema riconoscitivo è dato dagli automi, attraverso delle macchine, infatti, si può comprendere il linguaggio formale;

NOZIONI BASE NELLA TEORIA DEI LINGUAGGI FORMALI

SIMBOLO

Per simbolo, da vocabolario, s'intende un qualsiasi oggetto rappresentativo, solitamente inteso come segno grafico convenzionale unificato.

ALFABETO

Insieme finito di simboli (a_i), viene identificato con la lettera Σ (sigma)

$\Sigma = \{a_1, a_2, \ldots, a_k\}$ questa dicitura indica un alfabeto composto da k simboli

ESEMPIO ALFABETO:

Un esempio d'alfabeto è l'alfabeto binario, ossia un alfabeto costituito da solo due simboli, quali:

$\Sigma = \{a, b\}$ $\Sigma = \{0, 1\}$ $\Sigma = \{(,)\}$

PAROLA SU Σ

Detta anche "stringa", una parola è una sequenza finita di simboli presi su Σ (appartenenti a sigma). Per convenzione, la parola non contenente alcun simbolo, ossia la parola priva di simboli, prende il nome di "parola vuota" e viene indicata con ε (epsilon).

ESEMPIO PAROLA SULL'ALFABETO Σ:

Sia Σ = {a, b}
x = aababbab
y = babba
z = abbabaaaab
x, y, z sono parole su Σ

LUNGHEZZA DI W

La lunghezza di una parola identifica il numero di simboli, distinti per posizione, di cui si compone la parola stessa. La lunghezza di ε è zero. Data la parola w, indicheremo la lunghezza di w con le seguenti due diciture tra loro equivalenti: |w| oppure l(w).

ESEMPIO LUNGHEZZA DI UNA PAROLA SULL'ALFABETO Σ:

Sia Σ = {a, b}
x = aababbab |x| = 8
y = babba l(y) = 5
z = abbabaaaab |z| = 10

Σ* E Σ^+

Sono le due tipologie d'insiemi contenenti le parole costruibili con l'alfabeto Σ.

Σ*: insieme di tutte le parole costruibili sull'alfabeto Σ, includendo ε;

Σ^+: insieme di tutte le parole costruibili sull'alfabeto Σ, escludendo ε;

SE NE DERIVANO LE SEGUENTI FORMULE:

$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ # sigma star è uguale a sigma più con l'aggiunta della parola vuota (ε)
$\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ # sigma più è uguale a sigma star cui è stata tolta la parola vuota (ε)

PRODOTTO DI GIUSTAPPOSIZIONE

Detto anche "prodotto di concatenazione", il prodotto di giustapposizione è un'operazione binaria su Σ*, identificata con il simbolo "·", così definita:

$\forall x, y \in \Sigma^* \mid |x|, |y| \in N \qquad \exists z \mid z = x \cdot y$

date due parole (x, y), costruite sull'alfabeto Σ, le cui lunghezze sono due misure finite, possiamo asserire che esiste una terza parola (z) tale che z sia il risultato del prodotto di giustapposizione tra x e y.

ESEMPIO INTRODUTTIVO

Dati
$x = x_1x_2x_3\ldots x_n$ $|x| = n$
$y = y_1y_2y_3\ldots y_m$ $|y| = m$
$z = x \cdot y = x_1x_2x_3\ldots x_ny_1y_2y_3\ldots y_m$ $|z| = n + m$

OSSERVAZIONE

La lunghezza di z, dove z è il risultato del prodotto di giustapposizione delle parole x e y, non è altro che la somma delle lunghezze di x e y.

PROPRIETÀ :

- Associativa: $\forall x, y, z \in \Sigma^*$ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
- Elemento neutro : $\forall x \in \Sigma^* \exists e \in \Sigma^* \mid$ $e \cdot x = x = x \cdot e$ dove $e = \varepsilon$.

NON PROPRIETÀ:

L'operazione di prodotto di giustapposizione non gode della proprietà commutativa, infatti:

$\forall x, y \in \Sigma^* \mid x \neq y \neq \varepsilon \qquad x \cdot y \neq y \cdot x$

ESEMPIO DI NON COMMUTATIVITÀ DEL "·" SULL'ALFABETO Σ:

Dato Σ = {a, …, z}
x, y ∈ Σ* | x = sa
y = le
z = x · y = sale
k = y · x = lesa z ≠ k quindi non commutativo

MONOIDE LIBERO GENERATO DA Σ

Grazie alle due proprietà di cui gode l'operazione di giustapposizione, l'insieme Σ* con l'operazione "·" e "ε" costituisce un monoide, più precisamente la terna (Σ*, ·, ε) prende il nome di monoide libero generato da Σ, dove:

- Σ^* indica l'insieme delle parole costruibili sull'alfabeto Σ;
- "·" è l'operazione che mi permette, prese due parole appartenenti a Σ^*, facendone il prodotto, di ottenere, quale risultato dell'operazione, una parola appartenente a Σ^*;
- "ε" è l'elemento neutro di Σ^* rispetto all'operazione "·";

ESEMPIO D'ALFABETO UNITARIO E PAROLE COSTITUITE SU DETTO ALFABETO:

Sia Σ = {a}

Esempi di parole su Σ sono: a aaa ε aa

Le relative lunghezze sono: |a| = 1 l(aaa) = 3 |ε| = 0 l(aa) = 2

a · aaa = aaaa

|a · aaa| = 4 = 1 + 3 =|a| + |aaa|

ricordando che le diciture |x| e l(x) sono equivalenti

ESEMPIO D'ALFABETO BINARIO E PAROLE COSTITUITE SU DETTO ALFABETO:

L'alfabeto binario è un alfabeto costituito da solo due simboli, n'è un esempio

Σ = {0, 1}

Esempi di parole su Σ sono: 1 110 ε 1001

le relative lunghezze sono: |1| = 1 |110| = 3 |ε| = 0 |1001| = 4

110 · 1001 = 1101001

|110 · 1001| = 7 = 3 + 4 =|110| + |1001|

SOTTOPAROLE D'UNA PAROLA

FATTORE

Data la parola "w", si dice fattore una sequenza di simboli contenuta in "w", se "f" è un fattore di "w", allora scelte opportunamente due parole "h" e "j" (che possono anche essere ε) avremo: w = h ·f ·j.

OSSERVAZIONI:

- ε è un particolare fattore poiché lo è per qualsiasi parola di qualsiasi alfabeto
- Data una parola "w" esiste sempre un fattore particolare che è "w" stessa;

ESEMPIO DI FATTORE:

Prendiamo la parola w = "incalzare", alcuni esempi di fattori di w sono:

ε, "calza", "incalzare", "nc", "incalz", "are"

PREFISSO

Data la parola "w", si dice prefisso una sequenza di simboli che iniziano "w", se "p" è un prefisso di "w", allora scelta opportunamente una parola "j" (che può anche essere ε) avremo: w = p · j.

OSSERVAZIONI:

- ε è un particolare prefisso giacché lo è per qualsiasi parola di qualsiasi alfabeto;
- Data una parola "w" esiste sempre un prefisso particolare che è "w" stessa;
- Un prefisso è un particolare fattore "f" del tipo: w = h · f · j con h = ε.

ESEMPIO DI PREFISSO:

Prendiamo la parola w = "incalzare", qui di seguito sono elencati TUTTI i prefissi di w sono:

ε, "i", "in", "inc", "inca", "incal", "incalz", "incalza", "incalzar", "incalzare"

SUFFISSO

Data la parola "w", si dice suffisso una sequenza di simboli che terminano "w", se "s" è un suffisso di "w", allora scelta opportunamente una parola "h" (che può anche essere ε) avremo: w = h · s.

OSSERVAZIONI:

- ε è un particolare suffisso in quanto lo è per qualsiasi parola di qualsiasi alfabeto;
- Data una parola "w" esiste sempre un suffisso particolare che è "w" stessa;
- Un suffisso è un particolare fattore "f" del tipo: w = h · f · j con j = ε.

ESEMPIO DI SUFFISSO:

Prendiamo la parola w = "incalzare", qui di seguito sono elencati TUTTI i suffissi di w sono:

"ε", "e", "re", "are", "zare", "lzare", "alzare", "calzare", "ncalzare", "incalzare"

LINGUAGGI PARTICOLARI
LINGUAGGIO L SULL'ALFABETO Σ

È un insieme di parole di Σ*, ossia un qualunque sottoinsieme finito o infinito $L \subseteq \Sigma^*$, si distinguono quattro tipologie di linguaggi:

1. ∅: è il linguaggio vuoto, la sua cardinalità è zero, è il linguaggio privo di qualsiasi elemento;
2. {ε}: è il linguaggio di ε, la sua cardinalità è uno, è il linguaggio composto solo dalla parola vuota;
3. LINGUAGGIO FINITO: è il linguaggio a cardinalità finita, ossia composto da un numero finito d'elementi;

ESEMPIO DI LINGUAGGIO FINITO:

∅ è un linguaggio finito di cardinalità uguale a zero

ESEMPIO DI LINGUAGGIO FINITO:

{ε} è un linguaggio finito di cardinalità uguale ad uno

ESEMPIO DI LINGUAGGIO FINITO:

L = {ε, a, aa, aaa} è un linguaggio finito di cardinalità uguale a quattro

ESEMPIO DI LINGUAGGIO FINITO:

Le parole dell'italiano scritto sono le parole sull'alfabeto {a, b, …, z} elencate in un vocabolario della lingua italiana = {a, abaco, …, zuppo} è anche questo un linguaggio finito in quanto costituito da un numero finito di parole, enumerabili.

4. LINGUAGGIO INFINITO: è il linguaggio a cardinalità infinita, ossia composto da un numero d'elementi inenumerabile o ∞;

ESEMPIO DI LINGUAGGIO INFINITO:

{a}*, precisando che con questa scrittura s'intende l'insieme di tutte le parole che si possono costruire a partire dall'alfabeto costituito dal singolo simbolo 'a', bisogna considerare il fatto che l'insieme di parole costruibili a partire da un alfabeto unitario ha cardinalità infinita, infatti:

$\{a\}^* = \{\varepsilon, a, aa, aaa, aaaa, aaaaa, \ldots\} = \{\varepsilon, a^1, a^2, a^3, a^4, \ldots\}$

ESEMPIO DI LINGUAGGIO INFINITO:

Le espressioni aritmetiche su prodotto e somma costituiscono un altro esempio di linguaggio infinito, bisogna però prima vedere come fare per formalizzare le espressioni aritmetiche sotto forma di linguaggio formale: indichiamo con Σ l'alfabeto delle espressioni aritmetiche, quali sono i simboli che costituiscono Σ?

Σ = {0, 1, …, 9, *, +, (,)}, definiamo allora L come il linguaggio delle espressioni aritmetiche.

Regole:

- $0 \in L$, $1 \in L$, …, $9 \in L$
- Se $x \in L$ e $y \in L \rightarrow (x + y) \in L$, $(x * y) \in L$
- Nient'altro appartiene a L se non parole ottenute a partire dalle due regole appena citate, ossia l'unico modo per costruire espressioni aritmetiche è partire da x e y (ossia da due espressioni aritmetiche) e o sommarle tra di loro oppure effettuare il prodotto, ossia comporre una nuova espressione aritmetica.

Questa è la formalizzazione di un'espressione aritmetica sotto forma dialogativa.

Dato L appena definito, analizziamo l'appartenenza a L delle seguenti parole:

$(7 + 5) \in L$?	SI
$(17 + 5 (\in L$?	NO
$((3 * 2) + 9) \in L$?	SI

ESEMPIO DI LINGUAGGIO INFINITO:

Le espressioni booleane sulle operazioni ¬, ^,∨ costituiscono un ulteriore esempio di linguaggio infinito, bisogna però prima vedere come fare per formalizzare le espressioni booleane sotto forma di linguaggio formale: indichiamo con Σ l'alfabeto delle espressioni booleane, quali sono i simboli che costituiscono Σ?

Σ = {0, 1, ¬, ^, ∨ , (,)} definiamo allora L come il linguaggio delle espressioni booleane;

Regole:

- $0 \in L$, $1 \in L$
- Se $x \in L$ e $y \in L \rightarrow (x \wedge y) \in L$, $(x \vee y) \in L$, $(\neg x) \in L$
- Nient'altro appartiene a L se non parole ottenute a partire dalle due regole appena citate, ossia l'unico modo per costruire espressioni booleane è, partendo dalle costanti 0 e 1, operarle tra loro tramite or oppure tramite and o ancora tramite il not.

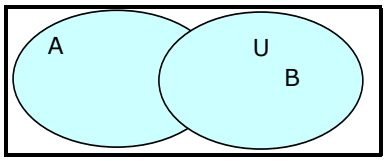
Dato L appena definito, analizziamo l'appartenenza a L delle seguenti parole:

$\neg(1 \wedge b) \in L$?	SI
$\neg(17 \wedge 5 (\in L$?	NO
$\neg((1 \wedge 0) (\in L$?	NO

OPERAZIONI SUI LINGUAGGI

Dato che i linguaggi sono sottoinsiemi di Σ^*, si possono applicare le usuali operazioni booleane:

UNIONE



Presi due linguaggi A e B su un certo alfabeto Σ (ossia A, B $\subseteq \Sigma^*$), si definisce unione di A e B un terzo linguaggio "U" contenente le parole che appartengono ad A o a B (o ad entrambi), la condizione è quindi che la parola appartenga almeno ad uno dei due linguaggi posti tra loro in unione. L'unione tra due linguaggi finiti è un linguaggio finito così definito:

$U = A \cup B = \{w \in \Sigma^* \mid w \in A \text{ oppure } w \in B\}$

OSSERVAZIONE: A e B sono sottoinsiemi di U, infatti, $A \subseteq U$, $B \subseteq U$.

ESEMPIO DI UNIONE

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

A = {w ∈ Σ* | prefisso w = 1} = 1 {0, 1}* # linguaggio le cui parole iniziano con il simbolo "1"

B = {w ∈ Σ* | suffisso w = 0} = {0, 1}* 0 # linguaggio le cui parole finiscono con il simbolo "0"

U = A ∪ B = {parole su Σ | prefisso = 1 oppure suffisso = 0} = 1 {0, 1}* oppure {0, 1}* 0

Prendiamo ora x, y, k, z ∈ Σ* :

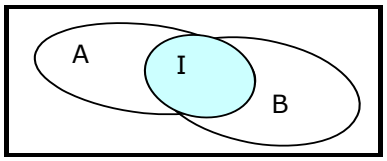
x = 1001 x ∈ U ? SI ha prefisso 1

y = 010 y ∈ U ? SI ha suffisso 0

k= 1010 k ∈ U ? SI ha entrambe le richieste valide

z = 011 z ∈ U ? NO, non ha né prefisso 1 né suffisso 0

INTERSEZIONE



Presi due linguaggi A e B su un certo alfabeto Σ (ossia A, B $\subseteq \Sigma^*$), si definisce intersezione di A e B un terzo linguaggio "I" contenente le parole che appartengono ad A e a B, la condizione è quindi che la parola appartenga ad entrambi i linguaggi. L'intersezione tra due linguaggi finiti è un linguaggio finito così definito:

$I = A \cap B = \{w \in \Sigma^* \mid w \in A \text{ e } w \in B\}$

OSSERVAZIONE: I è un sottoinsieme sia di A sia di B, infatti, $I \subseteq A$, $I \subseteq B$.

ESEMPIO DI INTERSEZIONE

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

A = 1 {0, 1}* # linguaggio le cui parole iniziano con il simbolo "1"

B = {0, 1}* 0 # linguaggio le cui parole finiscono con il simbolo "0"

I = A ∩ B = {parole su Σ | prefisso = 1 e suffisso = 0} = 1 {0, 1}* 0

Prendiamo ora x, y, k, z ∈ Σ* :

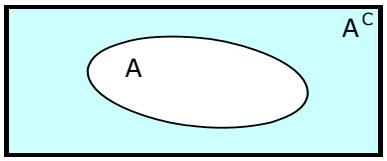
x = 1001 x ∈ I ? NO non ha suffisso 0

y = 010 y ∈ I ? NO non ha prefisso 1

k= 1010 k ∈ I ? SI ha entrambe le richieste valide

z = 011 z ∈ I ? NO

COMPLEMENTO



Preso il linguaggio A su un certo alfabeto Σ (ossia A $\subseteq \Sigma^*$), si definisce complemento del linguaggio A un secondo linguaggio, che solitamente viene denotato con "A^C", contenente tutte le parole di Σ^* che non appartengono al linguaggio A, di cui A^C è complemento, il complemento di un linguaggio finito è un linguaggio infinito ed è così definito:

$A^C = \{w \in \Sigma^* \mid w \notin A\}$

OSSERVAZIONE: A è disgiunto ad A^C, ossia $A \cap A^C = \varnothing$.

ESEMPIO DI COMPLEMENTO

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

A = 1 {0, 1}* # linguaggio le cui parole iniziano con il simbolo "1"

A^C = complemento di A = {parole su Σ | prefisso ≠ 1} = 0 {0, 1}*

Prendiamo ora x, y, k, z ∈ Σ* :

x = 1001 x ∈ A^C ? NO ha prefisso = 1

y = 010 y ∈ A^C ? SI ha prefisso ≠1

k= 1010 k ∈ A^C ? NO

z = 011 z ∈ A^C ? SI

PRODOTTO

Presi due linguaggi A e B su un certo alfabeto Σ (ossia A, B $\subseteq \Sigma^*$), si definisce prodotto un terzo linguaggio "P" contenente parole "w" tali che il prefisso di "w" è una parola del linguaggio A e la restante parte di "w" è una parola del linguaggio B, il prodotto tra due linguaggi finiti è ancora un linguaggio finito ed è così definito:

$AB = \{w \mid w = x \cdot y \text{ con } x \in A \text{ e } y \in B\}$

- Il prodotto tra linguaggi non è commutativo, infatti $AB \neq BA$

ECCEZIONE ALLA NON COMMUTATIVITÀ

Se prendiamo un alfabeto unitario, ossia $\Sigma \mid l(\Sigma) = 1$ e definiamo due linguaggi su Σ, allora il prodotto dei due linguaggi è commutativo, infatti:

sia $\Sigma = \{a\}$ sia $A \subseteq \Sigma^*$ e sia $B \subseteq \Sigma^*$

$AB = BA$

ESEMPIO DI PRODOTTO

Prendiamo due linguaggi A = {neutr, sal} B = {ino, one}

- Cos'è AB? Per ottenere il prodotto AB devo prendere tutte le parole appartenenti al linguaggio A e produrle con tutte le parole appartenenti al linguaggio B
  AB = {neutrino, neutrone, salino, salone}
- Cos'è BA? Per ottenere il prodotto BA devo prendere tutte le parole appartenenti al linguaggio B e produrle con tutte le parole appartenenti al linguaggio A
  BA = {inoneutr, inosal, oneneutr, onesal}

ESEMPIO DI PRODOTTO

RICORDANDO PRIMA CHE: {h}* indica il linguaggio formato da tutte le parole che si possono ottenere con l'utilizzo del simbolo 'h', ossia $\{\varepsilon, h, hh, hhh, \ldots\} = \{\varepsilon, h^1, h^2, h^3, \ldots\}$

A = {a}* = {ε, a, aa, aaa , …} B = {b}* = {ε, b, bb, bbb , …}

Cos'è AB?

AB = {w ∈ Σ*| w = xy dove x ∈ A e y ∈ B}

ε · ε = ε ε · b = b ε · bb = bb …

a · ε = a a · b = ab a · bb = abb …

aa · ε = aa aa · b = aab aa · bb = aabb …

…

$AB = \{\varepsilon, a^1, a^2, \ldots, a^i, \ldots, b^1, b^2, \ldots, b^j, \ldots, a^ib^j, \ldots\}$ dove i e j sono numeri arbitrari, dato che questo tipo di indicazione è troppo lunga si può riassumere con:

AB = a*b*, questa forma prende il nome di FORMA BREVE

POTENZA

Dato il linguaggio $A \subseteq \Sigma^*$, la potenza k-esima di A consiste nel produrre il A per se stesso k volte, ossia:

$$A^k = \underbrace{A \cdot A \cdot A \cdot \ldots \cdot A}_{k \text{ volte}}$$

la potenza k-esima di un linguaggio A può essere espressa anche in maniera ricorsiva, analogamente a come si procede per il fattoriale (n! = n * (n-1)!), infatti, $A^k = A \cdot A^{k-1}$

ESEMPIO DI POTENZA

L = {a}

- Cos'è L^k? Per ottenere la potenza L^k devo produrre il linguaggio L con se stesso k volte.
  Cosa significa produrre L per se stesso k volte? Significa produrre "a" per se stessa k volte ottenendo a^k.
  $L^k = a^k$
- Cos'è L^3? $L^3 = L \cdot L \cdot L = aaa = a^3$

ESEMPIO DI POTENZA

L = {a, b}

- Cos'è L^k? $L^k = \{ w \mid w \in \{a, b\}^* \text{ dove } |w| = k \}$
- Cos'è L^5? $L^5 = \{w \mid w \in \{a, b\}^* \text{ dove } |w| = 5 \}$
  Date le seguenti parole quali appartengono a L^5?
  "aacba" ? NO $\notin L^5$ infatti la parola "c" non è una parola di L
  "bbbbb" ? SI $\in L^5$
  "bbababb" ? NO $\notin L^5$ infatti | bbababb | = 7 ≠ 5
  "baaba" ? SI $\in L^5$

CHIUSURA DI KLEENE:

Dato un linguaggio L, la chiusura di Kleene si indica con L* o L^+ a seconda se:

L*, consiste nell'effettuare l'unione di tutti i linguaggi L^i con i che va da zero a infinito, ossia

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^k \cup \dots \cup L^\infty = \bigcup_{i=0}^{\infty} L^i$$

dove L^0 per convenzione è la parola vuota, ossia $L^0 = \{\varepsilon\}$

$L^* = \{w \mid w = x_1 \cdot x_2 \cdot \dots \cdot x_n \text{ dove } x_i \in L \text{ con } 1 \le i < n, n \ge 1\} \cup \{\varepsilon\}$

L^+, consiste nell'effettuare l'unione di tutti i linguaggi L^i con i che va da uno a infinito, escludendo quindi L^0

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \dots \cup L^k \cup \dots \cup L^\infty = \bigcup_{i=1}^{\infty} L^i$$

$L^+ = \{w \mid w = x_1 \cdot x_2 \cdot \dots \cdot x_n \text{ dove } x_i \in L \text{ con } 1 \le i < n, n \ge 1\}$

SE NE DERIVANO LE SEGUENTI FORMULE:

$L^* = L^+ \cup L^0$

$L^+ \neq L^* \setminus \{\varepsilon\}$

COME MAI $L^+ \neq L^* \setminus \{\varepsilon\}$?

In L^+ potrebbe esserci la parola vuota, derivante dal fatto che, se L = {ε, a}, ossia, se L è un linguaggio composto dalla parola vuota e dalla parola "a" allora tutti gli L^h conterranno ε, dire che $L^+ = L^* \setminus \{\varepsilon\}$ vorrebbe dire togliere a L* tutte le parole vuote, incluse quelle derivanti da L^1, L^2, … L^h, e non escludere solo L^0.

DIFFERENZA TRA L^n E L*:

L* è il linguaggio formato dalle parole ottenute moltiplicando, in tutte le permutazioni possibili, parole di L, unitamente alla parola ε. Si osserva che L* è un sottomonoide di Σ*, ed in particolare è il più piccolo fra i sottomonoidi di Σ* che contengono L.

L^n è il linguaggio formato dalle parole ottenute moltiplicando, in tutte le permutazioni possibili, parole di L prese "n" alla volta.

ESEMPIO DI CHIUSURA DI KLEENE

Sia L = {+1, -1} # linguaggio composto solo dalle parole "+1" e "-1", non contenente la parola vuota

- Cos'è L^+? $L^+ = L^1 \cup L^2 \cup \dots \cup L^k \cup \dots \cup L^\infty$

$L^1 = L = \{+1, -1\}$

$L^2 = L \cdot L = \{+1, -1\} \cdot \{+1, -1\} = \{+1+1, +1-1, -1+1, -1-1\}$

$L^3 = L \cdot L \cdot L = \{+1,-1\} \cdot \{+1,-1\} \cdot \{+1,-1\} = \{+1+1+1,+1+1-1,+1-1+1,+1-1-1,-1+1+1,-1+1-1,-1-1+1,-1-1-1\}$

…

$L^+ = \{+1, -1, +1+1, +1-1, -1+1, -1-1, +1+1+1, +1+1-1, \dots\}$ che sono i numeri interi (Z, i razionali)

- Cos'è L*? $L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^k \cup \dots \cup L^\infty = \{+1, -1\}^*$

ESEMPIO DI CHIUSURA DI KLEENE

Sia L = {a, bb} # linguaggio composto solo dalle parole "a" e "bb", non contenente ε

- Cos'è L^2? $L^2 = L \cdot L = \{a, bb\} \cdot \{a, bb\} = \{aa, abb, bba, bbbb\}$
- Cos'è L*? $L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^k \cup \dots \cup L^\infty = \{a, bb\}^*$
  ossia tutte le parole x tali che siano formate solo dai simboli "a" e "b" dove il simbolo "b" deve sempre comparire ripetuto due volte consecutivamente.

Date le seguenti parole x, y, k, z definire quali appartengono a L*

x = "aabba" ? SI ∈ L*

y = "bbabbabb"? NO ∉ L* infatti il simbolo "b" non è ripetuto due volte all'interno della parola

k = "aaabaaa" ? NO ∉ L* infatti il simbolo "b" non è ripetuto due volte all'interno della parola

z = "baaba" ? NO ∉ L* infatti il simbolo "b" è ripetuto due volte, ma non sono consecutive

ESEMPIO DI CHIUSURA DI KLEENE

Sia L = {bb} # linguaggio composto solo dalla parola "bb", non contenente ε

Cos'è L*? $L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^k \cup \dots \cup L^\infty = \{bb\}^*$

$L^* = \{w \mid w = (b)^{2k}, k \ge 0\}$ # parole di lunghezza pari

ESEMPIO DI CHIUSURA DI KLEENE

Sia L = {bb, b} # linguaggio composto solo dalle parole "b" e "bb", non contenente ε

Cos'è L*? $L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^k \cup \dots \cup L^\infty = b^*$

In questo caso abbiamo tutte le parole, sia di lunghezza pari sia di lunghezza dispari componibili con il simbolo 'b'

ESERCIZIO RIEPILOGATIVO

Sia $\Sigma = \{a\}$

siano A, B $\subseteq \Sigma$ così definiti:

$A = \{\varepsilon, a, aa, aaa\} = \{\varepsilon, a^1, a^2, a^3\}$

$B = \{aaa, aaaa, aaaaa\} = \{a^3, a^4, a^5\}$

Si calcolino le seguenti operazioni:

- $A \cup B$? $A \cup B = \{w \mid w \in A \text{ oppure } w \in B\} = \{\varepsilon, a^1, a^2, a^3, a^4, a^5\}$
- $A \cap B$? $A \cap B = \{w \mid w \in A \text{ e } w \in B\} = \{a^3\}$
- A^c ? $A^c = \{w \mid w \in \Sigma^* \text{ ma } w \notin A\} = \{a^4, a^5, a^6, \ldots, a^\infty\}$
- B^c ? $B^c = \{w \mid w \in \Sigma^* \text{ ma } w \notin B\} = \{\varepsilon, a^1, a^2, a^6, a^7, \ldots, a^\infty\}$
- AB? $AB = \{w \mid w = xy \text{ dove } x \in A \text{ e } y \in B\} = \{\varepsilon, a^1, a^2, a^3\} \cdot \{a^3, a^4, a^5\} =$
  $= \{\varepsilon a^3, \varepsilon a^4, \varepsilon a^5, a^1a^3, a^1a^4, a^1a^5, a^2a^3, a^2a^4, a^2a^5, a^3a^3, a^3a^4, a^3a^5\} =$
  $= \{a^3, a^4, a^5, a^4, a^5, a^6, a^5, a^6, a^7, a^6, a^7, a^8\} = \{a^3, a^4, a^5, a^6, a^7, a^8\}$
- BA? $BA = \{w \mid w = xy \text{ dove } x \in B \text{ e } y \in A\} = \{a^3, a^4, a^5\} \cdot \{\varepsilon, a^1, a^2, a^3\} =$
  $= \{a^3\varepsilon, a^3a^1, a^3a^2, a^3a^3, a^4\varepsilon, a^4a^1, a^4a^2, a^4a^3, a^5\varepsilon, a^5a^1, a^5a^2, a^5a^3\} =$
  $= \{a^3, a^4, a^5, a^6, a^4, a^5, a^6, a^7, a^5, a^6, a^7, a^8\} = \{a^3, a^4, a^5, a^6, a^7, a^8\}$
- A*? $A^* = A^0 \cup A^1 \cup A^2 \cup \ldots \cup A^k \cup \ldots \cup A^\infty = \{a\}^*$
- B*? $B^* = B^0 \cup B^1 \cup B^2 \cup \ldots \cup B^k \cup \ldots \cup B^\infty = \{\varepsilon, a^3, a^4, a^5, a^6, \ldots\}$
- $A^{+?}$ $A^+ = A^1 \cup A^2 \cup \ldots \cup A^k \cup \ldots \cup A^\infty = \{a\}^*$
- B^+ ? $B^+ = B^1 \cup B^2 \cup \ldots \cup B^k \cup \ldots \cup B^\infty = \{a^3, a^4, a^5, a^6, \ldots\}$

CODICE

Un linguaggio L si definisce codice, se ogni parola appartenente ad L^+ può essere decomposta in maniera univoca, come prodotto di parole di L. Il codice L possiede dunque l'importante proprietà di univoca decifrabilità, infatti, data una parola $w \in L^+$ (dove L è un codice) esiste uno e un solo modo di ottenere w come prodotto di parole di L. Le parole in L sono l'alfabeto del codice. L^+ costituisce l'insieme di tutti i messaggi che si possono scrivere con il codice. I codici NON contengono ε.

CODICI PREFISSI

Un linguaggio L in cui ogni parola non è prefisso di nessun'altra parola è chiaramente un codice, in cui ulteriormente ogni parola $w \in L^+$ può essere decodificata in linea leggendo da destra a sinistra, tali codici prendono il nome di codici prefissi o codici istantanei.

CODIFICA

Dato un alfabeto V e un codice $C \subseteq \Sigma^*$, una codifica di V in C è una corrispondenza biunivoca $f: V \to C$.
Si enuncerà che l'elemento $v \in V$ è codificato da f (v) e si scriverà v = f (v).

ESEMPIO DI CODICE PREFISSO

$L = \{aa, aba, baa\}$

Prendiamo una parola di L^+ e vediamo se è decomponibile in più modi: x = "baaabaaabaa", $x \in L^+$

decomposizione: $baa \cdot aba \cdot aa \cdot baa \in L^+$

Non sono possibili altre decomposizioni di x in L, quindi L è un codice, altresì L è un "codice prefisso" in quanto ciascuna parola non è prefisso di nessun'altra parola di L, ossia

- "aa" non è prefisso ne di "aba" ne di "baa"
- "aba" non è prefisso ne di "aa" ne di "baa"
- "baa" non è prefisso ne di "aa" ne di "aba"

ESEMPIO DI CODICE NON PREFISSO

$C = \{0, 01\}$

Prendiamo una parola di C^+ e vediamo se è decomponibile in più modi: y = "00010101", $y \in C^+$

decomposizione: $0 \cdot 0 \cdot 01 \cdot 01 \cdot 01 \in C^+$

Non sono possibili altre decomposizioni in C quindi è un codice.

la parola "0" $\in$ C è prefisso della parola "01" $\in$ C, quindi, nonostante sia un codice, non è un "codice prefisso"

ESEMPIO DI NON CODICE

$L = \{bab, aba, ab\}$

Prendiamo una parola di L^+ e vediamo se è decomponibile in più modi: k = "ababab", $k \in L^+$

decomposizione: $ab \cdot ab \cdot ab \in L^+$

$aba \cdot bab \in L^+$

sono possibili più decomposizioni quindi L non è un codice

ESEMPIO DI CODICE PREFISSO

Un altro esempio di codice è ASCII esteso, questo codice permette di codificare ogni carattere della tastiera come sequenza di 8 bit, i caratteri {…, 0, 1, …, 9, …, A, B, …, a, b, …, {, }, -, +, …} in ASCII = {00000000, 00000001, 00000010, …, 11111110, 11111111} = $\{0, 1\}^8$

Cos'è ASCII*? È una parola formata dai soli simboli '0' e '1' di lunghezza pari a 8k con k ≥ 1.

ESEMPIO DI NON CODICE

L = {bb, b}

Prendiamo una qualsiasi parola di L^+ e vediamo se è decomponibile in più modi: s = "bbb", s ∈ L^+

decomposizione: $b \cdot b \cdot b \in L^+$

$b \cdot bb \in L^+$

$bb \cdot b \in L^+$

sono possibili più decomposizioni quindi L non è un codice

RAPPRESENTAZIONE DI LINGUAGGI

Un linguaggio L è un insieme di parole su un dato alfabeto Σ.

come è possibile dare una descrizione di L? Le modalità con cui un linguaggio può essere descritto sono due e sono conseguenti alla tipologia di cardinalità del linguaggio:

1. Rappresentazione di linguaggi a cardinalità finita;
2. Rappresentazione di linguaggi a cardinalità infinita;

RAPPRESENTAZIONE DI LINGUAGGI FINITI

Se L è un LINGUAGGIO FINITO, ovvero composto da un numero finito di elementi, allora il linguaggio L può essere rappresentato estensivamente, mediante quindi l'elencazione di tutti gli elementi (parole) che compongono il linguaggio in oggetto.

ESEMPIO INTRODUTTIVO DI RAPPRESENTAZIONE ESTENSIVA

L = $\{w_1, w_2, \ldots, w_n\}$ dove $w_i \in L$, n è un numero finito;

RAPPRESENTAZIONE DI LINGUAGGI INFINITI

Se L è un LINGUAGGIO INFINITO, ovvero composto da un numero infinito o inenumerabile di elementi, allora il linguaggio L deve essere rappresentato intensivamente, attraverso quindi una proprietà P(w) descrivibile con una quantità finita di informazioni, che risulta vera su tutte e sole le parole del linguaggio.

L = {w | P(w) = vero}, in altri termini, P esprime cosa deve soddisfare w per appartenere ad L;

a loro volta i linguaggi infiniti, se ammettono una descrizione (non è detto, infatti, che tutti i linguaggi ammettano la descrizione mediante un sistema), possono essere descritti con due sistemi diversi:

- SISTEMI RICONOSCITIVI permettono di stabilire, data una parola, se essa appartiene o meno al linguaggio. I linguaggi che ammettono un sistema riconoscitivo sono detti "linguaggi ricorsivi" o "decidibili", i più comuni sistemi riconoscitivi sono gli "automi a stati finiti" e "automi a pila", capaci di riconoscere due sottoclassi di linguaggi ricorsivi, detti rispettivamente "linguaggi regolari" e "linguaggi liberi da contesto".
- SISTEMI GENERATIVI permettono di costruire parole del linguaggio, i più comuni sistemi generativi sono le grammatiche;

SISTEMI RICONOSCITIVI

Un sistema riconoscitivo per il linguaggio L ⊆ Σ*, è in grado di calcolare, mediante un algoritmo che prende il nome di "RICONOSCITORE", la funzione caratteristica del linguaggio.

FUNZIONE CARATTERISTICA

È la funzione che ci permette di individuare, fornita una parola in ingresso, se la parola appartiene o meno al linguaggio L. Viene indicata con il simbolo $X_L(x)$ che si legge "ki di L", prende come input il valore x (parola da controllare) e restituisce come output:

$$X_L(X) = \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases}$$

ESEMPIO DI SISTEMA RICONOSCITIVO

Si consideri come riconoscitore il parser di un compilatore per linguaggio di programmazione C, il parser è un algoritmo che, ricevendo in ingresso una stringa "w", stabilisce se w è la codifica ASCII di un programma in C sintatticamente corretto, in tal caso, il compilatore fornisce quale output il codice oggetto del programma w, altrimenti elenca gli opportuni messaggi di errore. Ovviamente l'algoritmo del parser è sviluppato in modo tale che se come output ritorna 1 (le parole appartengono al linguaggio C), allora deve restituire il codice oggetto, se come output ritorna 0 (le parole non appartengono al linguaggio C), allora deve inviare messaggi di errore.

PROCEDURA, ALGORITMO, PROGRAMMA E INTERPRETE

PROCEDURA

Prende il nome di procedura una sequenza finita di istruzioni che possono essere eseguite automaticamente e che possono portare ad un risultato se il programma (che deve essere scritto bene e deve seguire le direttive previste dalla procedura) è progettato per fornirlo.

ALGORITMO

Prende il nome di algoritmo una procedura che termina sempre, qualsiasi sia l'input fornitogli in ingresso, un algoritmo non prevede quindi situazioni di loop infiniti o di break;

PROGRAMMA

Prende il nome di programma lo strumento mediante il quale vengono definite le procedure e gli algoritmi.
I programmi hanno quattro connotazioni (quattro diversi punti di vista che li descrivono):

- SINTATTICO: un programma è una parola $w \in \{0, 1\}^*$ ossia una sequenza di bit, questa connotazione fornisce la descrizione del programma w mediante la sua codifica ASCII, un programma viene quindi inteso sintatticamente come un insieme di parole su Σ^* dove Σ^* è l'alfabeto binario formato solo da "0" e "1";
- SEMANTICO: un programma "w" è una procedura che, opportunamente inizializzata, genera una sequenza di passi di calcolo che possono terminare fornendo un risultato, indichiamo con F_w la semantica del programma w. Se "w" sintatticamente è una parola binaria che corrisponde al codice ASCII di un programma e "x" è sintatticamente una parola binaria che viene passata come dato input al programma, indicheremo con $F_w(x)$ il risultato dell'esecuzione del programma w calcolato sull'input x.

Le altre due connotazioni secondo le quali possono essere descritti i programmi interessano il comportamento che i programmi assumono durante la loro esecuzione:

- NON TERMINANO, sono programmi che vanno in loop (entrano in un ciclo continuo, ripetitivo), il risultato dell'esecuzione di questa tipologia di programma viene indicato con $F_w(x)\uparrow$, il quale indica che il programma w su input x non termina la sua esecuzione.

ESEMPIO DI PROGRAMMA CHE NON TERMINA

Consideriamo il programma w che accetta in input un numero "x" e poi calcola il più grande numero primo maggiore di x. Qualsiasi sia l'input fornito in ingresso al programma w, detto programma non terminerà mai la sua esecuzione in quanto i numeri primi sono infiniti, infatti, dato un qualsiasi numero "x" in input al programma, esistono sempre infiniti numeri primi maggiori di x e dato che il programma deve trovare il più grande numero primo maggiore di x, questo proseguirà la sua ricerca senza riuscire ad uscire dal ciclo.
Il risultato del programma w è quindi $F_w(x)\uparrow$.

- TERMINANO IN 0/1, sono programmi che terminano la loro esecuzione fornendo come output solo due tipologie di valori: o 0 oppure 1, il risultato dell'esecuzione di questa tipologia di programma viene indicato con $F_w(x)\downarrow$, il quale indica che il programma w su input x termina e, se il risultato dell'esecuzione è stato 0, scriveremo $F_w(x) = 0$, nel caso contrario, scriveremo $F_w(x) = 1$.

ESEMPIO DI PROGRAMMA CHE TERMINA SEMPRE

Consideriamo un programma w che preso in input un numero binario "x" deve restituire in uscita se il numero inserito è pari o dispari. Qualsiasi sia l'input fornito in ingresso al programma w, detto programma sarà sempre in grado di terminare la sua esecuzione in quanto la determinazione della parità o disparità di un numero binario è possibile mediante l'analisi dell'ultima cifra del numero, se questa risulterà essere "0" allora x è pari, se, contrariamente, l'ultima cifra risulterà essere "1" allora x è dispari.
Il risultato del programma w è quindi $F_w(x)\downarrow$

INTEPRETE

Indicato con u, l'interprete è un programma che prende in input un altro programma w ed un dato x, fornendo in output il risultato che ottiene simulando l'esecuzione del programma w sul dato x, ossia l'interprete fornisce in output $F_w(x)$, se w è un programma, in caso contrario ritornerà quale uscita un indefinito ($\perp$).

- INPUT DI U: $x, w \in \{0, 1\}^*$ dove x è un dato e w è un programma;
- ESECUZIONE DI U: l'interprete, avendo come ingresso la parola x$w, simula l'esecuzione della procedura codificata con il programma w su ingresso del dato x, detta simulazione viene rappresentata dalla dicitura: $F_u(x, w)$ o $F_u(x \$ w)$
- OUTPUT DI U: il risultato dell'interprete può assumere due valori distinti:

$$F_u(x, w) = \begin{cases} F_w(x) & \text{sse w è un programma} \\ \uparrow & \text{sse w NON è un programma} \end{cases}$$

Se w è un programma allora l'interprete termina e fornisce come risultato Fw(x), in caso contrario, ossia se w non è un programma l'esecuzione dell'interprete non termina.

CLASSIFICAZIONE DEI LINGUAGGI

RICORSIVO

Un linguaggio L si dice ricorsivo (o deducibile) se esiste un algoritmo implementato dal programma w tale che, passandogli in input un dato $x \in \{0, 1\}^*$:

$$\begin{cases} F_w(x) = 1 & \text{sse } x \in L \\ F_w(x) = 0 & \text{sse } x \notin L \end{cases}$$

Se L è un linguaggio ricorsivo allora ho un programma w che termina sempre, in quanto implementa un algoritmo, e, come abbiamo visto prima nella definizione di algoritmo, l'algoritmo mi garantisce sempre che il programma restituirà un risultato dalla sua esecuzione.

Un linguaggio ricorsivo può essere specificato con una quantità finita di informazioni; essa è data dal testo w del programma che calcola la funzione caratteristica di L, un linguaggio ricorsivo è quindi un linguaggio che ammette un sistema riconoscitivo.

Inoltre, dato $x \in \Sigma^*$, risulta possibile decidere se $x \in L$ semplicemente eseguendo il programma w su ingresso x e recuperando il risultato dopo un numero finito di passi di calcolo.

RICORSIVAMENTE NUMERABILE

Un linguaggio L si dice ricorsivamente numerabile (o semideducibile) se esiste una procedura implementata dal programma w tale che, passandogli in input un dato $x \in \{0 ,1\}^*$:

$$\begin{cases} F_w(x) = 1 & \text{sse } x \in L \\ F_w(x)\uparrow & \text{sse } x \notin L \end{cases}$$

Se L è un linguaggio ricorsivamente numerabile allora ho un programma w che non termina sempre, infatti implementa una procedura, e, come abbiamo visto prima nella definizione di procedura, la procedura non mi garantisce la terminazione del programma.

Un linguaggio ricorsivamente numerabile può essere specificato con una quantità finita di informazioni; essa è data dal testo w del programma che risponde 1 quando $x \in L$ e non termina l'esecuzione quando $x \notin L$. contrariamente a quanto accade per i linguaggi ricorsivi, nei linguaggi ricorsivamente numerabili, quando $x \notin L$ non è possibile recuperare l'informazione in un numero finito di passi di calcolo, se ne conclude quindi che un linguaggio ricorsivamente numerabile non ammette un sistema riconoscitivo bensì solo un sistema generativo.

Inoltre, dato $x \in \Sigma^*$, non risulta sempre possibile decidere se $x \in L$ semplicemente eseguendo il programma w su ingresso x e recuperando il risultato, infatti, se $x \in L$ il programma, dopo un numero finito di passi di calcolo, fornirà in uscita il risultato, in caso contrario, se $x \notin L$, il programma entrerà in loop senza fornire alcun risultato. Pensare di definire che $x \notin L$ se, dopo un ragionevole quantitativo di tempo, non si ottiene alcun risultato dall'esecuzione del programma è errato, infatti, non viene garantita la non appartenenza di x ad L in quanto il programma potrebbe ancora essere in esecuzione per la creazione del risultato.

TEORIA DEI LINGUAGGI

LINGUAGGIO = PROBLEMA

Sappiamo che linguaggio è sinonimo di problema, quindi, studiare un linguaggio significa studiare e risolvere un problema, infatti, associato a ciascun linguaggio esiste un problema, che indicheremo con la dicitura P_L (problema associato al linguaggio L da un'opportuna codifica che solitamente è il codice ASCII), in generale infatti un linguaggio L può essere descritto mediante la seguente proprietà:

$L = \{w \mid P(w) = 1\}$ ossia, il linguaggio L è costituito da quelle parole w tali che soddisfino la proprietà P, dove dire che w soddisfa P è analogo a scrivere P(w) = 1.

- Se L è un linguaggio ricorsivo, il problema P_L associato a detto linguaggio prende il nome di decidibile, ossia è un problema risolvibile interamente in maniera automatica.
  L ricorsivo $\leftrightarrow$ P_L decidibile
- Se L è un linguaggio ricorsivamente numerabile, il problema P_L associato a detto linguaggio prende il nome di semidecidibile, ossia è un problema risolvibile in maniera automatica solo parzialmente.
  L ricorsivamente numerabile $\leftrightarrow$ P_L semidecidibile

P_L è così definito:

- in input abbiamo delle parole binarie, $w \in \{0, 1\}^*$
- in output abbiamo la risposta SI/NO associata alla seguente domanda: "w soddisfa P?"

dato un problema, P_L, codifico ogni istanza del problema P_L in un'istanza binaria e poi mi pongo la domanda, ossia creo un algoritmo A per P_L, dove A è un algoritmo che stabilisce se $w \in L$ restituendo 1 o 0 in base all'appartenenza o non appartenenza della parola w al linguaggio L.

Se $w \in L$ allora, dato che $L = \{w \mid P(w) = 1\}$ w soddisfa anche P_L

Se $w \notin L$ allora, dato che $L = \{w \mid P(w) = 1\}$ w non soddisfa neanche P_L

L LINGUAGGIO RICORSIVO ⇒ L^C LINGUAGGIO RICORSIVO?

Se L è un linguaggio ricorsivo, allora esiste un algoritmo implementato dal programma A tale che, passandogli in input un dato $x \in \{0, 1\}^*$:

$$\begin{cases} F_A(x) = 1 & \text{sse } x \in L \\ F_A(x) = 0 & \text{sse } x \notin L \end{cases}$$

cosa significa che L^C è il complemento del linguaggio L?

$L^C = \{w \in \Sigma^* \mid w \notin L\}$

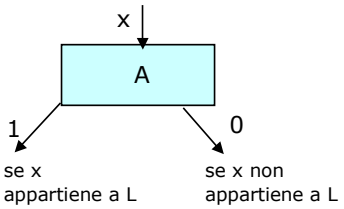
si necessita quindi di un secondo programma B tale che, passandogli in input un dato $x \in \{0, 1\}^*$:

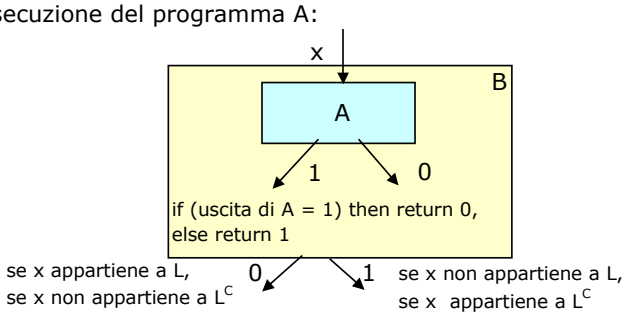
$$\begin{cases} F_B(x) = 0 & \text{sse } x \in L \\ F_B(x) = 1 & \text{sse } x \notin L \end{cases}$$

Dato che, se $x \in L$, $F_A(x) = 1$ e, se $x \notin L$, $F_A(x) = 0$, allora possiamo pensare che B è un programma che, passandogli in input il risultato per programma A ($F_A(x)$) si comporta nella seguente maniera:

$$\begin{cases} F_B(x) = 0 & \text{sse } F_A(x) = 1 \\ F_B(x) = 1 & \text{sse } F_A(x) = 0 \end{cases}$$

Vediamo ora in dettaglio i due programmi A e B e le modifiche che si necessitano di operare per consentire al programma B di lavorare sul risultato dell'esecuzione del programma A:





se ne deduce che, anche il programma B, implementa un algoritmo, e come abbiamo visto dalla definizione di linguaggio ricorsivo, sappiamo che se il programma che definisce il linguaggio implementa un algoritmo, allora il linguaggio in oggetto è ricorsivo, abbiamo quindi dimostrato che se L è ricorsivo, allora L^C è anch'esso un linguaggio ricorsivo.

RICORSIVO = RICORSIVAMENTE NUMERABILE?

Per poter dimostrare che ricorsivo = ricorsivamente numerabile necessitiamo di dimostrare due punti distinti:

A. ricorsivo ⊆ ricorsivamente numerabile
B. ricorsivamente numerabile ⊆ ricorsivo

A. RICORSIVO ⊆ RICORSIVAMENTE NUMERABILE?

Sia L un linguaggio ricorsivo, L è anche un linguaggio ricorsivamente numerabile, infatti,

Se L è un linguaggio ricorsivo, allora ∃ il programma w tale che, passandogli in input un dato $x \in \{0, 1\}^*$:

$$\begin{cases} F_w(x) = 1 & \text{sse } x \in L \\ F_w(x) = 0 & \text{sse } x \notin L \end{cases}$$

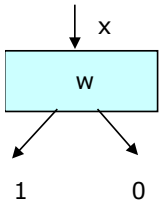
costruiamo ora una procedura P che simula l'algoritmo w e, se il risultato dell'esecuzione di w è 0 ($F_w(x) = 0$), genera una computazione che non termina (loop). Questo è verificabile in quanto un algoritmo può essere sempre peggiorato al fine di non farlo terminare, trasformandosi così in una procedura.

TEOREMA

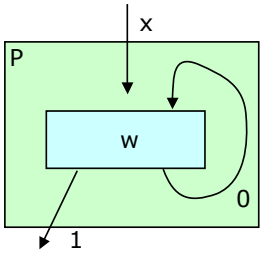
Se L è un linguaggio ricorsivo ⇒ L è un linguaggio ricorsivamente numerabile

DIMOSTRAZIONE

Se L è un linguaggio ricorsivo → per definizione ∃ w t.c.:



posso ora costruire una procedura P, basata sull'algoritmo w, tale che, se x non appartiene al linguaggio L ($F_w(x) = 0$) allora entra in un loop



in questo modo ho trasformato un algoritmo in una procedura e, dato che, un algoritmo implementa linguaggi ricorsivi e, una procedura implementa linguaggi ricorsivamente numerabili, con detta trasformazione di w in P è stato modificata la classificazione di L da linguaggio ricorsivo a linguaggio ricorsivamente numerabile

La dimostrazione appena effettuata ha permesso di individuare che i linguaggi ricorsivi sono anche linguaggi ricorsivamente numerabili a patto di un peggioramento dell'algoritmo che consente di trasformarlo in una procedura, quindi: linguaggio ricorsivo ⊆ linguaggio ricorsivamente numerabile

Dopo aver dimostrato che un linguaggio ricorsivo è un linguaggio ricorsivamente numerabile, per poter definire che le due classificazioni sono uguali dobbiamo dimostrare anche il punto B del teorema, ossia dimostrare che un linguaggio ricorsivamente numerabile è anche un linguaggio ricorsivo.

B. RICORSIVAMENTE NUMERABILE ⊆ RICORSIVO?

Al fine di ottenere questa tipologia di dimostrazione ci serviamo del disegno sottostante, ossia cercheremo di dimostrare l'esistenza di un linguaggio D che è un linguaggio ricorsivamente numerabile ma non un linguaggio ricorsivo, nel caso in cui detta dimostrazione non sia possibile allora avrò dimostrato che i linguaggi ricorsivamente numerabili sono linguaggi ricorsivi e quindi ricorsivo = ricorsivamente numerabile, in caso contrario, la dimostrazione dell'esistenza di un linguaggio D che è ricorsivamente numerabile ma non ricorsivo dimostrerà che le due classificazioni dei linguaggi sono distinte, ossia ricorsivamente numerabile ≠ ricorsivo;

Per effettuare la dimostrazione necessitiamo innanzitutto di definire: il linguaggio D e il suo complemento, D^C, dove D^C sarà un linguaggio che non si può trattare neanche parzialmente.

DEFINIZIONE DI D:

$D = \{x \in \{0, 1\}^* \mid F_u(x, x)\downarrow\}$

Data una parola x, binaria, quindi interpretabile dall'interprete, sia come dato che come programma, la passo quale input all'interprete. Per poter asserire che $x \in D$, il programma eseguito dall'interprete deve terminare.

Input dell'interprete u: $x \in \{0, 1\}^*$ è un programma

Output dell'interprete: Se $x \in D$ → $F_u(x, x) = F_x(x)$

il concetto è di dare in input al programma x se stesso.

DEFINIZIONE DI D^C:

$D^C = \{ x \in \{0, 1\}^* \mid F_u(x, x)\uparrow \}$

Data una parola x quale input all'interprete u, per poter asserire che $x \notin D$, u non dovrà terminare.

Input dell'interprete u: $x \in \{0, 1\}^*$ è un programma

Output dell'interprete: Se $x \notin D$ → $F_u(x, x) \uparrow$ (ossia non termina)

TEOREMA

Proviamo l'esistenza di un linguaggio D così definito:

1. D è ricorsivamente numerabile
2. D non è ricorsivo
3. D^C non è ricorsivamente numerabile

DIMOSTRAZIONE

DIMOSTRAZIONE DEL PUNTO 1)

Necessitiamo di dimostrare che D è un linguaggio ricorsivamente numerabile, per effettuare questa dimostrazione abbiamo bisogno di una procedura chiamata RICNUM:

```
        RICNUM:
1       Richiede in input x ∈ {0, 1}*
2       Chiama l'interprete U e gli passa l'argomento (x $ x)
            U:
3           Prende in input l'argomento (x $ x)
4           Esegue Fu (x, x)
5               Se x ∈ D → Fu (x, x) ↓, Fu (x, x) = 1 viene ritornato a RICNUM
6               Se x ∉ D → Fu (x, x) ↑, la funzione non termina, non si torna a RICNUM
7       Output = 1 sse riceve da Fu (x, x) il valore 1
```

ANALISI DESCRITTIVA DELLA PROCEDURA RICNUM:

- Prende in input la parola $x \in \{0, 1\}^*$;
- Richiama l'interprete U;
- Passa all'interprete x $ x quali input;
- Rimane in attesa del risultato dell'interprete;
- Una volta ottenuto il risultato, se questo è uguale ad "1" allora restituisce quale output "1" .

RICNUM prende il nome di "microistruzione" in quanto al suo interno richiama un interprete.

ANALISI DESCRITTIVA DELL'ESECUZIONE DELL'INTERPRETE:

- Prende in input x $ x, dove x ∈ {0, 1}*;
- Esegue la sua funzione ossia Fu (x, x) che come abbiamo visto da definizione di D e D^C può dare due tipologie di uscire, quindi ora i casi saranno due:
  - se x ∈ D → F_u (x, x)↓, F_u (x, x) = 1 la procedura implementata dall'interprete termina e restituisce "1";
  - se x ∉ D → F_u (x, x)↑, la procedura implementata dall'interprete non termina e, dato che persiste in uno stato di loop, non restituisce output

CASO I: TEST DI RICNUM PASSANDO IN INPUT X ∈ D

Passiamo a RICNUM il dato x ∈ {0, 1}*, dove x ∈ D

1 RICNUM: riceve in input x ∈ {0, 1}*
2 chiama l'interprete U e gli passa l'argomento (x $ x)
3 U: prende in input l'argomento (x $ x)
4 esegue F_u (x, x)
5 x ∈ D → F_u (x, x)↓, F_u (x, x) = 1 viene ritornato a RICNUM
7 output = 1

la procedura RICNUM ha terminato e ha restituito in output il valore "1"

CASO II: TEST DI RICNUM PASSANDO IN INPUT X ∉ D

Passiamo a RICNUM il dato x ∈ {0, 1}*, dove x ∉ D

1 RICNUM: riceve in input x ∈ {0, 1}*
2 chiama l'interprete U e gli passa l'argomento (x $ x)
3 U: prende in input l'argomento (x $ x)
4 esegue F_u (x, x)
6 x ∉ D → F_u (x, x)↑, la funzione non termina, non si torna a RICNUM

la procedura RICNUM non ha terminato quindi è in situazione di loop, infatti, è ancora in attesa di un risultato da parte di U.

Il corretto funzionamento della procedura RICNUM ci consente di identificare D come un linguaggio ricorsivamente numerabile, ricordiamo infatti che se un linguaggio è descrivibile mediante una procedura questo prende il nome di linguaggio semidecidibile o ricorsivamente numerabile.

DIMOSTRAZIONE DEL PUNTO 2)

Necessitiamo di dimostrare che D non è un linguaggio ricorsivo, per effettuare questa dimostrazione abbiamo bisogno dell'utilizzo della tecnica dell'assurdo, ossia nel dimostrare che D è un linguaggio ricorsivo dobbiamo ottenere un assurdo il quale ci permetterà di asserire che l'ipotesi iniziale è errata, per operare questa dimostrazione abbiamo bisogno di una procedura chiamata ASSURDOA.

IPOTESI INIZIALE: D È RICORSIVO

ASSURDOA:
1 Richiede in input x ∈ {0, 1}*
2 Chiama la funzione $APPARTENENZA_D$ e gli passa come argomento (x)
APPARTENENZA$_D$:
3 Prende in input l'argomento x ∈ {0, 1}*
4 Esegue y = $F_{APPARTENENZAD}$ (x)
5 Se x ∈ D → $F_{APPARTENENZAD}$ (x) ↓, y = 1
6 Se x ∉ D → $F_{APPARTENENZAD}$ (x) ↓, y= 0
7 Output = y
8 If(y == 1) then Output: "1-F_u (x, x) "
9 Else Output : " 0 "

ANALISI DESCRITTIVA DELLA PROCEDURA ASSURDOA:

- Prende in input la parola x ∈ {0, 1}*;
- Richiama la funzione $APPARTENENZA_D$;
- Passa alla funzione x quale input;
- Rimane in attesa del risultato della funzione;
- Una volta ottenuto il risultato lo memorizza nella variabile y;
- Effettua il test y == 1, (ossia x ∈ D):
  1. se vero → restituisce "1 – f_u(x, x)";
  2. se falso → restituisce "0" ;

ASSURDOA termina sia con x ∈ D, (y = 1) sia con x ∉ D (y ≠ 1), per essere implementata, necessita di un linguaggio ricorsivo, D infatti è ricorsivo per ipotesi.

ANALISI DESCRITTIVA DELLA FUNZIONE APPARTENENZA$_D$

- Prende in input la parola $x \in \{0, 1\}^*$;
- Esegue la sua funzione ossia $F_{APPARTENENZAD}$ (x) che può dare due tipologie di uscite:
  1. se $x \in D \rightarrow F_{APPARTENENZAD}$ (x) $\downarrow$, la funzione termina e restituisce "1";
  2. se $x \notin D \rightarrow F_{APPARTENENZAD}$ (x) $\downarrow$, la funzione termina e restituisce "0";

APPARTENENZA$_D$ termina sempre, necessita quindi di un linguaggio ricorsivo (D ricorsivo per ipotesi).

COSA SIGNIFICA IMPLEMENTARE ASSURDOA? implementare la procedura ASSURDOA significa eseguire il programma 'e' che la definisce; come abbiamo visto da definizione, infatti, procedure e algoritmi vengono sempre definiti mediante programmi, si avrebbe quindi:

$F_e(x)$ = ASSURDOA(x)

Ossia l'esecuzione del programma 'e' sull'input x, dato che 'e' è il programma che implementa ASSURDOA, coincide con l'esecuzione della procedura ASSURDOA su input x. Per procedere con la dimostrazione si necessita della scelta di un input x da passare alla procedura, dato che x è una collezione di simboli binari, e, analogamente, anche un programma è una collezione di simboli binari, posso decidere di passare quale input della procedura ASSURDOA proprio il programma 'e' che la definisce, effettuando quindi: ASSURDOA (e).

COS'È ASSURDOA(e)? Per definizione di programma sappiamo che:

F_e (x) = ASSURDOA (x) $\Rightarrow$ **(a)** $F_e(e)$ = ASSURDOA(e)

COSA SIGNIFICA EFFETTUARE $F_e(e)$? dalla definizione di interprete sappiamo che:

$F_w(x) = F_u(x, w)$

Dove: $F_w(x)$ # è il risultato dell'esecuzione della funzione F su input x operata dal programma w;
$F_u(x, w)$ # è il risultato dell'esecuzione della funzione F operata dall'interprete u, il quale fornisce in input il dato x al programma w;

Quindi riportando questa uguaglianza ad $F_e(e)$ otteniamo che:

$F_w(x) = F_u(x, w)$ $\Rightarrow$ **(b)** $F_e(e) = F_u(e, e)$

Ossia l'interprete u prende il programma 'e' e lo esegue passandogli quale input il programma 'e'.

UGUAGLIANZA COMPLETA (1):

- (a) = F_e (e) = ASSURDOA (e)
- (b) = F_e (e) = F_u (e, e)
- (1) = (a) + (b) = **F_e (e) = ASSURDOA (e) = F_u (e, e)**

CASO I: TEST DI ASSURDOA PASSANDO IN INPUT E $\in$ D

Passiamo a ASSURDOA il programma $e \in \{0, 1\}^*$, dove $e \in D$

1 ASSURDOA: riceve in input $e \in \{0, 1\}^*$
2 Chiama la funzione APPARTENENZA$_D$ e gli passa come argomento (e)
3 APPARTENENZA$_D$: Prende in input il programma $e \in \{0, 1\}^*$
4 Esegue y = $F_{APPARTENENZAD}$ (e)
5 $e \in D \rightarrow F_{APPARTENENZAD}$ (e) $\downarrow$, y = 1
7 Output = 1
8 y == 1 then Output: "1-F_u (e, e) "

La procedura ASSURDOA ha terminato, vediamo che valore ha assunto mediante l'uguaglianza (1):

F_e (e) = ASSURDOA (e) = F_u (e, e) $\Rightarrow$ 1 - F_u (e, e) = ASSURDOA (e) = F_u (e, e)

Sappiamo che Fu(e, e) può assumere due valori: 1 / 0, testiamo entrambi i sottocasi:

SOTTOCASO 1: Fu(e, e) = 1

1 - F_u(e,e) = ASSURDOA(e) = F_u(e,e) $\Rightarrow$ 1 – 1 = ASSURDOA(e) = 1 $\Rightarrow$ 0 = ASSURDOA(e) = 1 $\Rightarrow$ ASSURDO

SOTTOCASO 2: Fu(e, e) = 0

1 - F_u(e,e) = ASSURDOA(e) = F_u(e,e) $\Rightarrow$ 1 – 0 = ASSURDOA(e) = 0 $\Rightarrow$ 1 = ASSURDOA(e) = 0 $\Rightarrow$ ASSURDO

A) abbiamo quindi dedotto che 'e' NON può appartenere a D in quanto da assurdo

CASO II: TEST DI ASSURDOA PASSANDO IN INPUT E $\notin$ D

Passiamo a ASSURDOA il programma $e \in \{0, 1\}^*$, dove $e \notin D$

1 ASSURDOA: riceve in input $e \in \{0, 1\}^*$
2 Chiama la funzione APPARTENENZA$_D$ e gli passa come argomento (e)
3 APPARTENENZA$_D$: Prende in input il programma $e \in \{0, 1\}^*$
4 Esegue y = $F_{APPARTENENZAD}$ (e)
6 $e \notin D \rightarrow F_{APPARTENENZAD}$ (e) $\downarrow$, y = 0
7 Output = 0
9 $y \neq 1$ then Output: "0"

La procedura ASSURDOA ha terminato, vediamo che valore ha assunto mediante l'uguaglianza (1):

F_u (e, e) = ASSURDOA (e) = 0

Sappiamo che Fu(e, e) ↑(non termina), otteniamo quindi: $\perp$ = ASSURDOA (e) = 0 ASSURDO

B) abbiamo quindi dedotto che 'e' NON può NON appartenere a D in quanto da assurdo

DEDUZIONE FINALE:

effettuiamo l'unione delle due deduzioni ricavate rispettivamente dai casi I e II, ossia uniamo la soluzione A) con la soluzione B), otteniamo:

A) 'e' NON può appartenere a D
B) 'e' NON può NON appartenere a D
} NON ESISTE 'e'

Siamo all'ASSURDO finale in quanto per ipotesi avevamo D linguaggio ricorsivo e quindi ∃ 'e', la non esistenza di 'e' implica che l'ipotesi fosse FALSA, si ottiene quindi che D NON è ricorsivo.

DIMOSTRAZIONE DEL PUNTO 3)

Necessitiamo di dimostrare che D^C non è un linguaggio ricorsivamente numerabile, ossia che D^C è un linguaggio legato ad un problema che non è trattabile in maniera automatica dal calcolatore neanche parzialmente. Anche in questo caso utilizziamo la tecnica dell'assurdo, nel dimostrare che D^C è un linguaggio ricorsivamente numerabile dobbiamo ottenere un assurdo.

IPOTESI INIZIALE: D^C È RICORSIVAMENTE NUMERABILE

QUALI SONO LE INFORMAZIONI E GLI ELEMENTI CHE ABBIAMO?

A) D è ricorsivamente numerabile, in quanto dimostrato al punto 1) del teorema, quindi esiste una procedura che chiameremo 'z' (in realtà sappiamo essere RICNUM), così definita:

z: Richiede in input x ∈ {0, 1}*
Chiama l'interprete U e gli passa l'argomento (x $ x)
U: Prende in input l'argomento (x $ x)
Esegue F_u (x, x)
Se x ∈ D → F_u (x, x) ↓, F_u (x, x) = 1 viene ritornato a z
Se x ∉ D → F_u (x, x) ↑, la funzione non termina, non si torna a z
Output = 1 sse riceve da Fu (x, x) il valore 1

B) D^C è ricorsivamente numerabile, per ipotesi iniziale del punto 3) del teorema, quindi esiste una procedura che chiameremo 'y' (in realtà sappiamo essere RICNUM), così definita:

y: Richiede in input x ∈ {0, 1}*
Chiama l'interprete U e gli passa l'argomento (x $ x)
U: Prende in input l'argomento (x $ x)
Esegue F_u (x, x)
Se x ∈ D^C → F_u (x, x) ↓, F_u (x, x) = 1 viene ritornato a y
Se x ∉ D^C → F_u (x, x) ↑, la funzione non termina, non si torna a y
Output = 1 sse riceve da Fu (x, x) il valore 1

COSA SIGNIFICA x ∈ D^C?

Per definizione di complemento se x ∈ D^C → x ∉ D
e viceversa se x ∈ D → x ∉ D^C

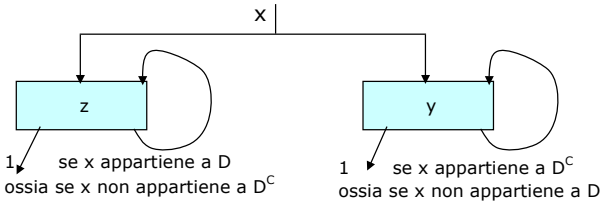
COMPLETEZZA DELLE PROCEDURE

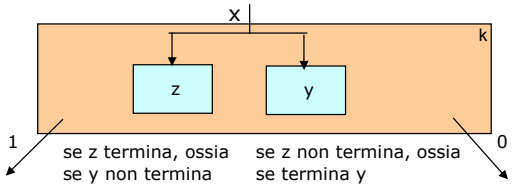
Dalla definizione di complemento deduciamo che:

z ↓ (termina) ↔ x ∈ D viceversa y ↓ (termina) ↔ x ∉ D

Costruisco ora una terza procedura k, la quale, passa in ingresso x ad entrambe le procedure e esegue il test:

if (z ↓) then return (1)
else return (0)





la procedura 'k' è quindi un programma che si comporta nella seguente maniera:

se x ∈ D return 1
se x ∉ D return 0

implementa quindi un algoritmo che risolve il riconoscimento del linguaggio D, ma se ho un algoritmo allora so che D è un linguaggio ricorsivo, dato che al punto 2) del teorema ho dimostrato che D NON è ricorsivo ottengo ASSURDO, deduciamo quindi che l'ipotesi è errata, quindi D^C non può essere ricorsivamente numerabile.

DIMOSTRAZIONE DEL PUNTO 3)

Abbiamo appena dimostrato che D^C non è un linguaggio ricorsivamente numerabile, ossia che D^C è un linguaggio legato ad un problema che non è trattabile in maniera automatica dal calcolatore neanche parzialmente.
Per effettuare questa dimostrazione abbiamo effettuato la parallelizzazione di due procedure, abbiamo, infatti, supposto che le due procedure 'z' e 'y' potessero lavorare simultaneamente senza provocare problemi.
Dimostriamo ora il punto 3) senza l'utilizzo di parallelizzazioni, ossia nel dimostrare che D^C è un linguaggio ricorsivamente numerabile dobbiamo ottenere un assurdo il quale ci farà asserire quindi che NON è ricorsivamente numerabile, per effettuare questa dimostrazione, senza l'utilizzo di procedure parallele, necessitiamo di una procedura chiamata ASSURDOB

IPOTESI INIZIALE: D^C È RICORSIVAMENTE NUMERABILE

```
ASSURDOB:
Richiede in input x ∈ {0, 1}*
k = 0
h= z
        while ( z↑ and y↑)
        {
                if (h == z)   then    h = y
                              else    h = z
                esegui h per k passi
                k = k+1
        }
if (z↓) then   output: "1"
        else   output: "0"
```

ANALISI DESCRITTIVA DELLA PROCEDURA ASSURDOB:

- Prende in input la parola x (binaria, ossia x ∈ {0, 1}*)
- Inizializza una variabile k che fungerà da contatore per i passi di esecuzione delle procedure y e z, la variabile inizialmente viene inizializzata a zero
- Inizializza una terza procedura 'h' cui assegna la procedura z
- Effettua un ciclo di while
- Se termina z allora la procedura restituisce 1, contrariamente restituisce 0

ANALISI DESCRITTIVA DEL CICLO WHILE:

- Il ciclo while consente di operare su entrambe le procedure (z e y) in maniera non parallela, infatti, ogni qual volta viene interrotta l'esecuzione di una delle due procedure, per k passi, inizia l'esecuzione dell'altra procedura per k+1 passi. Questo viene effettuato finché non termina una delle due procedure
- USCITA DAL WHILE:

  0 = non termina 1 = termina

z	y	while	
0	0	0	
0	1	1	
1	0	1	
1	1	X	indeterminata, non possono terminare contemporaneamente

COME VENGONO ALTERNATE LE DUE ESECUZIONI, DI "y" E "z"?

k = 0 h = z → esecuzione passo 0 di h
Blocco di z, parte y
k = 1 h = y → esecuzione passo 0, 1 di h
Blocco di y, parte z
k = 2 h = z → esecuzione passo 0, 1, 2 di h
Blocco di z, parte y
k = 3 h = y → esecuzione passo 0, 1, 2, 3 di h
Blocco di y, parte z
k = 4 h = z → esecuzione passo 0, 1, 2, 3, 4 di h
…

DEDUZIONE FINALE:

L'ultima istruzione, ossia il test "if" mostra l'esistenza di un algoritmo che termina sempre, quindi impone che D sia ricorsivo, il quale per dimostrazione del punto 2) è falso, ottenendo l'ASSURDO.

SISTEMI GENERATIVI

Un altro modo per rappresentare linguaggi è quello generativo; introduciamo questa nozione in relazione al concetto di sistema formale o calcolo logico. In particolare in questo corso introdurremo un importante modello di sistema generativo: la grammatica.

Motiviamo il concetto di calcolo logico a partire da una problematica che in qualche misura tutti conoscono: la geometria elementare nel piano. In trattazione formale della geometria si può seguire questo approccio:

1. vengono per prima cosa introdotti vari nomi (piano, punto , linea, retta, ecc.) mediante i quali sarà possibile comporre un insieme infinito di affermazioni, rappresentabili da un linguaggio L = {f: f affermazione sulla geometria elementare}. Esempi di tali affermazioni sono le seguenti frasi (viste come parole del linguaggio): le diagonali di un rombo sono perpendicolari, i tre lati di un triangolo sono paralleli, …
2. si interpretano le affermazioni avendo come modello il piano, i suoi punte e le sue rette. Alcune delle affermazioni in L risultano vere (esempio: le diagonali di un rombo sono perpendicolari), altre false (esempio: i tre lati di un triangolo sono paralleli). In lingua di principio, è possibile quindi considerare il linguaggio G $\subseteq$ L formato dalle affermazioni vere: G = {t: t affermazione vera della geometria elementare}.

Un calcolo logico deve permettere di "dimostrare" tutte e sole le affermazioni vere, date nel linguaggio G. le dimostrazioni dovranno a loro volta poter essere descritte da parole di un linguaggio D, così da dar senso (vero/falso) alla frase: d è la dimostrazione di f

è ragionevole richiedere che:

a. deve essere possibile verificare in un numero finito di passi se d è la dimostrazione di f oppure no.
b. Se d è la dimostrazione di f, allora f deve essere un'affermazione vera nella geometria elementare (correttezza del calcolo: si possono dedurre solo affermazioni vere)
c. Se t è un'affermazione vera della geometria elementare, allora deve esistere una sua dimostrazione d (completezza del calcolo: ogni affermazione vera è dimostrabile)

CALCOLO LOGICO

Un calcolo logico è dato da una funzione V: $\Sigma^* \times U^* \rightarrow \{0, 1\}$ calcolabile da un algoritmo A.

```
A:
Richiede in input x ∈ {0, 1}*
Richiede in input d ∈ {0, 1}*
Esegue V(x, d)
        If:
        {
                If (V(x, d) == 1)       then y = 1
                                        Else y = 0
                Return y
        }
output: y
```

ANALISI DESCRITTIVA DELL'ALGORITMO A:

- prende in input la parola x (binaria, ossia x $\in$ {0, 1}*)
- prende in input la parola d (binaria, ossia d $\in$ {0, 1}*)
- esegue la funzione V(x, d)
- fornisce il risultato dell'esecuzione della funzione V all'istruzione if, rimanendo in attesa del risultato dell'istruzione
- fornisce quale output il risultato di IF

ANALISI DESCRITTIVA DELL'ISTRUZIONE IF:

- l'istruzione if consente di operare un test il quale, assegna a "y" (dove y è una variabile binaria, y $\in$ {0, 1}*) il valore 1 se V(x, d) = 1, assegna a y il valore 0 se V(x, d) = 0.
- Restituisce y all'algoritmo A

dove U* denota l'insieme delle potenziali "dimostrazioni". Se V(x, d) = 1 si dirà che "d è una dimostrazione (o testimone) di x". L'esistenza dell'algoritmo A soddisfa la richiesta che sia possibile verificare in un numero finito di passi se d è la dimostrazione di x oppure no.

CALCOLO LOGICO CORRETTO, sia L $\subseteq \Sigma^*$ un linguaggio, il calcolo logico V è detto corretto per L se V(x, d) = 1 implica x $\in$ L, questa condizione viene detta correttezza del calcolo perché, se x ammette una dimostrazione d nel calcolo, allora x deve appartenere a L.

CALCOLO LOGICO COMPLETO, sia $L \subseteq \Sigma^*$ un linguaggio, ogni qual volta $x \in L$, allora $\exists\ d \in U^*$ per cui $V(x, d) = 1$ questa condizione viene detta completezza del calcolo perché garantisce che, quando $x \in L$, è sempre possibile trovarne una dimostrazione d nel calcolo.

CALCOLO LOGICO PER L

Sia $L \subseteq \Sigma^*$ un linguaggio, si definisce un calcolo logico per L un calcolo logico V corretto e completo per L, in tal caso sarà:

$L = \{x \mid \exists\ d \quad V(x, d) = 1\}$

L AMMETTE CALCOLO LOGICO ↔ L RICORSIVAMENTE NUMERABILE

Non tutti i linguaggi ammettono un calcolo logico; mostriamo qui che i linguaggi che ammettono calcoli logici sono tutti e soli i linguaggi ricorsivamente numerabili.

A tal riguardo, consideriamo per prima cosa una corrispondenza biunivoca tra U* e l'insieme dei numeri naturali $N = \{x \mid 1, 2, 3, \ldots\}$. Una corrispondenza naturale è per esempio quella indotta dall'ordine lessicografico, così ottenuta:

1. si ordinano per prima cosa i simboli di U
2. date $x, y \in U^*$, sarà $x \leq y$ se la lunghezza di x è minore della lunghezza di y, oppure, nel caso di ugual misura delle lunghezze, x precede y nell'ordine alfabetico.

Le parole in U* risultano ordinate totalmente dalla relazione ≤. Indicheremo con d_1 la prima parola, d_2 la seconda, …, d_n la n-esima parola di U*.

OSSERVAZIONE: ricordiamo che si parla di relazione d'ordine quando una relazione R su un insieme X soddisfa le seguenti proprietà:

- riflessiva: $\forall\ a \in X,\ a\ R\ a$ #qualsiasi elemento di X è in relazione con se stesso
- antisimmetrica: $\forall\ a, b \in X$ se $a\ R\ b$ e $b\ R\ a \rightarrow a = b$ #presi due elementi a, b appartenenti all'insieme X, se a è in relazione con b e b è in relazione con a, allora devono essere uguali;
- transitiva: $\forall\ a, b, c \in X$ se $a\ R\ b$ e $b\ R\ c \rightarrow a\ R\ c$ #presi tre elementi a, b, c dell'insieme X, se a è in relazione con b e b è in relazione con c, allora anche a deve essere in relazione con c;

ricordiamo altresì che si parla di relazione d'ordine totalmente ordinata se, presi due elementi qualsiasi di X, essi sono confrontabili, ossia $\forall\ a, b \in X$ è possibile stabilire se a R b oppure b R a.

TEOREMA:

Per poter dimostrare che L ammette un calcolo logico ↔ L è ricorsivamente numerabile necessitiamo di dimostrare due punti distinti:

A. Se L ammette un calcolo logico ⇒ L è ricorsivamente numerabile
B. Se L è ricorsivamente numerabile ⇒ ammette un calcolo logico

DIMOSTRAZIONE:

A. SE L AMMETTE UN CALCOLO LOGICO ⇒ L È RICORSIVAMENTE NUMERABILE

Per procedere con la dimostrazione dobbiamo esibire una procedura "L" che sulle parole del linguaggio $L \subseteq \Sigma^*$ restituisca 1, mentre sulle parole non appartenenti a L non termini. Per effettuare detta dimostrazione so di essere in possesso di un calcolo logico, ossia posso calcolare la funzione V(w, d)

```
L:
Richiede in input x ∈ Σ*
inizializza n = 1
        while ( V(x, dn) = 0)
        {
                n = n + 1
        }
Output = 1
```

ANALISI DESCRITTIVA DELLA PROCEDURA L:

- prende in input la parola $x \in \Sigma^*$
- inizializza un contatore n, il valore iniziale di n è 1
- esegue il ciclo while e rimane in attesa dell'uscita dal ciclo
- fornisce quale output 1 se il ciclo while viene terminato

ANALISI DESCRITTIVA DEL CICLO WHILE:

- il ciclo while consente di iterare tra i diversi d_i nella ricerca di $d \mid V(x, d) = 1$
- l'uscita dal while è assicurata solo nel caso in cui $V(x, d_n) = 1$, ma, se $x \in L$, $\exists\ n \mid V(x, d_n) = 1$

se ne deduce che la procedura L è in grado di terminare il ciclo while e quindi di poter fornire quale output 1 solo se $x \in L$, infatti in caso contrario, se $x \notin L$ per ogni n vale che $V(x, d_n) = 0$, ossia non si avrà la possibilità di terminare il ciclo while, ne segue che L è ricorsivamente numerabile in quanto la procedura L si comporta analogamente alla procedura RICNUM (vedi pg. 13) la quale definisce L come linguaggio ricorsivamente numerabile.

B. SE L È RICORSIVAMENTE NUMERABILE ⇒ L AMMETTE CALCOLO LOGICO

Supponiamo che esista una procedura w per cui $F_w(x) = 1$ per ogni $x \in L$, mentre $F_w(x) \uparrow$ per ogni $x \notin L$. costruiamo il seguente calcolo logico legato e dipendente dalla procedura w appena definita:
$V(x, d_n) = 1$ se w ↓ in n passi
$V(x, d_n) = 0$ se w ↑
Se $x \in L$, ∃ n per cui la procedura w su ingresso x termina in n passi, e quindi esiste la dimostrazione $d = d_n$ tale che $V(x, d) = 1$. se invece $x \notin L$, la procedura w su ingresso x non termina e quindi ∀ n si ha $V(x, d_n) = 0$.
Il risultato appena ottenuto apre al seguente scenario, dato un linguaggio L, si hanno i seguenti due casi:

- L è ricorsivamente numerabile ⇒ ∃ un calcolo logico per L, corretto e completo
- L non è ricorsivamente numerabile ⇒ qualsiasi calcolo logico V che è corretto per L è necessariamente incompleto. È quindi possibile trovare una parola $x \in L$ | non esista una dimostrazione dell'appartenenza mediante calcolo logico in quanto $V(x, d_n) = 0$ per ogni d.

X PROBLEMA DI HILBERT

La scoperta di linguaggi di interesse logico non ricorsivamente numerabili è il nucleo dei risultati di incompletezza trovati da Goedel intorno al 1930. presentiamo qui un esempio basato sul X problema di Hilbert, uno dei 23 problemi aperti proposti da Hilbert durante il Congresso di Matematica di Parigi nel 1900. tale problema richiede di trovare tecniche risolutive per le equazioni diofantee (ED), ed in particolare di determinare un metodo per decidere se, dato un polinomio a coefficienti interi $p(x_1, x_2, x_3, \ldots, x_n)$, l'equazione $p(x_1, x_2, x_3, \ldots, x_n) = 0$ ammette una soluzione sugli interi.
Il problema può essere descritto in maniera formale considerando il linguaggio:
ED = { ∃ xi con i che va da 1 a n dove $p(x_1, x_2, \ldots, x_n) = 0$ | $p(x_1, x_2, \ldots, x_n)$ sia un polin. a coefficienti interi}
Fissata la struttura Z = <{…, -2, -1, 0, 1, 2, …}, +, *> degli interi relativi con somma e prodotto, l'affermazione $\exists x_1 \exists x_2 \exists x_3 \ldots \exists x_n$ dove $p(x_1, x_2, \ldots, x_n) = 0$ risulta vera se esistono interi $a_1, a_2, \ldots, a_n$ tali che $p(a_1, a_2, \ldots, a_n) = 0$

ESEMPIO

L'affermazione $\exists x_1 \exists x_2 |(6x_1 - 10x_2 = 5)$ risulta falsa. Qualsiasi intero a_i moltiplicato per un numero pari da sempre come risultato un numero pari ($6a_1$ è sempre pari, $10a_2$ è sempre pari), la differenza tra due numeri pari è sempre un numero pari ($6a_1 - 10a_2$ è sempre pari), concludendo, dato che 5 è un numero dispari, non esistono a_1, a_2 | $|6a_1 - 10a_2 = 5$.
L'affermazione $\exists x_1 \exists x_2 (x_1^4 - 5x_2^2 = 1)$ risulta invece vera, come è dimostrato dagli interi $x_1 = 3$, $x_2 = 4$:
$3^4 - 5*4^2 = 81 - 5*16 = 81 - 80 = 1$
L'insieme delle affermazioni in ED che, interpretate sugli interi, risultano vere, forma un linguaggio $X \subset ED$ così definito: X = { $\exists x_1 \exists x_2 \ldots \exists x_n$ dove $p(x_1, x_2, \ldots, x_n) = 0$ | ∃ interi $a_1, a_2, \ldots, a_n$ | $p(a_1, a_2, \ldots, a_n) = 0$ }
Andiamo ora a classificare il linguaggio X:

- X è ricorsivamente numerabile? SI, Poiché l'affermazione $\exists x_1 \exists x_2 \ldots \exists x_n$ dove $p(x_1, x_2, \ldots, x_n) = 0$ è vera se e solo se si possono trovare interi $a_1, a_2, \ldots, a_n$ | $p(a_1, a_2, \ldots, a_n) = 0$, un calcolo logico V per X può essere dato da:
  $V(\exists x_1 \exists x_2 \ldots \exists x_n\ p(x_1, x_2, \ldots, x_n) = 0, a_1, a_2, \ldots, a_n) = 1$, se $p(a_1, a_2, \ldots, a_n) = 0$
  $V(\exists x_1 \exists x_2 \ldots \exists x_n\ p(x_1, x_2, \ldots, x_n) = 0, a_1, a_2, \ldots, a_n) = 0$, se $p(a_1, a_2, \ldots, a_n) \neq 0$
- X è ricorsivo? NO, Il celebre risultato ottenuto nel 1967 da Matiyasevich (e di cui non possiamo qui dare la dimostrazione), ci consente di asserire che: X non è ricorsivo. La non ricorsività di X preclude dunque la possibilità di trovare un algoritmo per la soluzione delle equazioni diofantee, rispondendo in modo negativo al X Problema di Hilbert.
- Indaghiamo ora un nuovo problema, che chiameremo Ineguaglianze Diofantee (ID), collegato al precedente.
  ID chiede di decidere se, dato un polinomio $p(x_1, x_2, \ldots, x_n)$ a coefficienti interi, risulta $p(a_1, a_2, \ldots, a_n) \neq 0$ per tutti gli interi $a_1, a_2, \ldots, a_n$. Esso è descritto dal linguaggio Y dove:
  Y = { $\forall x_1 \forall x_2 \ldots \forall x_n$ dove $p(x_1, x_2, \ldots, x_n) \neq 0$ | $p(a_1, a_2, \ldots, a_n) \neq 0$ per tutti gli interi $a_1, a_2, \ldots, a_n$ }
  Il linguaggio Y può essere classificato come segue:

- Y è ricorsivamente numerabile? NO, Si osservi che l'affermazione $\forall x_1 \forall x_2 \dots \forall x_n$ dove $p(x_1, x_2, \dots, x_n) \neq 0$ è vera se e solo se l'affermazione $\exists x_1 \exists x_2 \dots \exists x_n\ p(x_1, x_2, \dots, x_n) = 0$ è falsa. Quindi:
  $\forall x_1 \forall x_2 \dots \forall x_n\ p(x_1, x_2, \dots, x_n) \neq 0 \in Y \leftrightarrow \exists x_1 \exists x_2 \dots \exists x_n\ p(x_1, x_2, \dots, x_n) = 0 \notin X$
  dato che sappiamo che se un elemento non appartiene ad un linguaggio L appartiene al suo complemento L^C, avremo :
  $\exists x_1 \exists x_2 \dots \exists x_n\ p(x_1, x_2, \dots, x_n) = 0 \notin X \leftrightarrow \exists x_1 \exists x_2 \dots \exists x_n\ p(x_1, x_2, \dots, x_n) = 0 \in X^C$
  dalla completezza delle due istruzioni avremmo :
  $\forall x_1 \forall x_2 \dots \forall x_n\ p(x_1, x_2, \dots, x_n) \neq 0 \in Y \leftrightarrow \exists x_1 \exists x_2 \dots \exists x_n\ p(x_1, x_2, \dots, x_n) = 0 \in X^C$
  se y fosse un linguaggio ricorsivamente numerabile allora anche X^C lo sarebbe, ma come abbiamo appena dimostrato X è ricorsivamente numerabile, essendo ora per dimostrazione sia X che X^C due linguaggi ricorsivamente numerabili, X sarebbe ricorsivo, ricordiamo infatti che se X e X^C sono classificabili all'interno della stessa classe allora sono ricorsivi (vedi pagina 12 "L = LINGUAGGIO RICORSIVO $\Rightarrow$ L^C LINGUAGGIO RICORSIVO), ma X ricorsivo si scontra con il teorema di Matiyasevich, concludendo:

Per quanto visto, Y non può ammettere nessun calcolo logico che sia corretto e completo. In particolare, preso un qualsiasi calcolo logico V corretto per Y, allora tale calcolo è necessariamente incompleto. Questo significa che esiste una affermazione $\forall x_1 \forall x_2 \dots \forall x_n\ p(x_1, x_2, \dots, x_n) \neq 0$ tale che:

1. L'affermazione $\forall x_1 \forall x_2 \dots \forall x_n\ p(x_1, x_2, \dots, x_n) \neq 0$ è vera, e cioè risulta che $p(a_1, a_2, \dots, a_n) \neq 0$ per tutti gli interi $a_1, a_2, \dots, a_n$
2. La stessa affermazione $\forall x_1 \forall x_2 \dots \forall x_n\ p(x_1, x_2, \dots, x_n) \neq 0$ non ammette tuttavia una dimostrazione in V, cioè vale che $V(\forall x_1 \forall x_2 \dots \forall x_n\ p(x_1, x_2, \dots, x_n) \neq 0, d) = 0$ per ogni dimostrazione d.

Problemi più generali, la cui soluzione non è fornita da una semplice risposta booleana, possono essere modellati mediante funzioni da interi in interi. Il dominio di una funzione rappresenta l'insieme delle istanze del problema; le immagini rappresentano le soluzioni. Indichiamo con F una funzione che codifica un particolare problema P' che ha soluzione algoritmica. Un possibile algoritmo per il calcolo di F(x), il cui valore equivale alla soluzione di P' su istanza x, può essere progettato usando un riconoscitore per il linguaggio $L_F = \{a^n b^{F(n)} \mid n \geq 0\}$, questo è possibile in quanto è sufficiente interrogare il riconoscitore su parole della forma $a^x b^y$, con $y \geq 0$, finché non si ottiene una risposta affermativa.

LE GRAMMATICHE

Come abbiamo visto in precedenza, i linguaggi ricorsivamente numerabili ammettono sistemi generativi, uno degli esempi più significativi di sistema generativo è dato dalle grammatiche.

ESEMPIO INTRODUTTIVO N° 1 (E.I. N°1)

Consideriamo il seguente insieme di regole descritte informalmente, che consentono la generazione di alcune frasi dell'italiano:

- una <frase> è la giustapposizione di un <soggetto>, di un <predicato> e di un <complemento>;
- un <soggetto> è la giustapposizione di un <articolo> e di un <nome>;
- un <complemento> è la giustapposizione di un <articolo> e di un <nome>;
- un <articolo> è "il";
- un <nome> è "cane", "gatto" o "topo";
- un <predicato> è "mangia" o "teme".

Esempi di frasi generate sono "il gatto mangia il topo" o "il cane teme il gatto". Osserviamo che tali frasi sono parole sull'alfabeto {il, cane, gatto, topo, mangia, teme}, quindi le regole precedenti denotano un linguaggio $L \subseteq \{il, cane, gatto, topo, mangia, teme\}^*$, mentre i simboli <frase>, <soggetto>, <complemento>, <predicato>, <articolo>, <nome> sono utilizzati per generare tale linguaggio, ma non fan parte di parole del linguaggio.

Le regole di produzione di cui gode detto linguaggio sono:

<frase> → <soggetto> <predicato> <complemento>
<soggetto> → <articolo> <nome>
<complemento> → <articolo> <nome>
<articolo> → il
<nome> → cane | gatto | topo
<predicato> → mangia | teme

ESEMPIO INTRODUTTIVO N° 2 (E.I. N°2)

Consideriamo il seguente insieme di regole descritte informalmente, che consentono la generazione dei messaggi vocali che vengono diffusi, nelle stazioni delle ferrovie dello stato, mediante gli altoparlanti i quali forniscono informazioni in merito ad orari di partenza, ritardi, binari e altro ancora, detti messaggi ci appaiono interrotti o con modifica di tonalità di voce, questo a causa del fatto che sono messaggi registrati, analizzeremo qui di seguito i messaggi forniti per i ritardi dei treni:

- un <messaggio> è la giustapposizione delle seguenti informazioni: <treno> da <città> viaggia con <tempo> di ritardo;
- un <tempo> è la giustapposizione di una <quantità> e di una <unità>;
- un <treno> è "regionale", "interregionale" o "eurostar";
- una <città> è "Milano", "Torino", "Roma", "Rimini", "Foggia" o "Napoli";
- una <quantità> è "1", "2", "3", "4",…, "60";
- una <unità> è "ore" o "minuti".

esempi di frasi generate sono "regionale da Milano viaggia con 2 ore di ritardo" o "eurostar da Napoli viaggia con 45 minuti di ritardo" o ancora "interregionale da Roma viaggia con 3 ore di ritardo". Osserviamo che tali frasi sono parole sull'alfabeto { regionale, interregionale, eurostar, Milano, Torino, Roma, Rimini, Foggia, Napoli, 1, 2, …, 60, minuti, ore }, quindi le regole precedenti denotano un linguaggio $L \subseteq$ { regionale, interregionale, eurostar, Milano, Torino, Roma, Rimini, Foggia, Napoli, 1, 2, …, 60, minuti, ore }*, mentre i simboli <messaggio>, <treno>, <città>, <tempo>, <quantità>, <unità> sono utilizzati per generare tale linguaggio, ma non fan parte di parole del linguaggio.

Le regole di produzione di cui gode detto linguaggio sono:

<messaggio> → <treno> da <città> viaggia con <tempo> di ritardo
<tempo> → <quantità> <unità>
<treno> → regionale | interregionale | eurostar
<città> → Milano | Torino | Roma | Rimini | Foggia | Napoli
<quantità> → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | … | 59 | 60
<unità> → ore | minuti

ALCUNE NOTAZIONI

METASIMBOLO, o anche detto "variabile" o "simbolo non terminale", viene posto all'interno dei tag "<", ">", costituisce una parte di testo da sostituire con simboli terminali;
in E.I. n° 1 i metasimboli sono: <frase>, <soggetto>, <complemento>, <predicato>, <articolo>, <nome>.
In E.I. n° 2 i metasimboli sono: <messaggio>, <treno>, <città>, <tempo>, <quantità>, <unità>.

SIMBOLO TERMINALE, o anche detto "simbolo", costituisce la parte fissa di testo da non sostituire;
in E.I. n° 1 i simboli terminali sono: il, cane, gatto, topo, mangia, teme
in E.I. n° 2 i simboli terminali sono: regionale, interregionale, eurostar, Milano, Torino, Roma, Rimini, Foggia, Napoli, 1, 2, …, 60, minuti, ore..

PAROLA, sequenza di simboli terminali, se w è una parola $w \in \Sigma^*$

FORMA SENTENZIALE, sequenza di simboli terminali e metasimboli, se f è una forma sentenziale, $f \in (M \cup \Sigma)^*$

ASSIOMA o anche detto "simbolo iniziale", è il metasimbolo di partenza
in E.I. n° 1 l'assioma è: frase
in E.I. n° 2 l'assioma è: messaggio

GRAMMATICA, sistema definito mediante la seguente quadrupla di elementi: $G = <\Sigma, M, P, S>$, dove:

- G è la grammatica;
- Σ è l'alfabeto dei simboli terminali (il quale deve essere un insieme finito);
- M è l'alfabeto dei metasimboli (anch'esso è un insieme finito ed è disgiunto da Σ, dove ricordiamo che per insiemi disgiunti si intende che l'intersezione dei due insiemi fornisce insieme vuoto, ossia $A \cap B = \varnothing$);
- P è l'insieme di regole di produzione (anche questo è un insieme finito);
- S è un elemento in M detto assioma o simbolo di partenza.

Descrivere una grammatica significa definire i quattro elementi sopra citati.

REGOLA DI PRODUZIONE, è una regola di scrittura della forma $\alpha \rightarrow \beta$, con

- $\alpha \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
- $\beta \in (\Sigma \cup M)^*$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M inclusa ε

L'applicazione di una regola di produzione del tipo $\alpha \rightarrow \beta$ ad una parola w dove $w = x\,\alpha\,y$ produce la parola z dove $z = x\,\beta\,y$, quest'ultima è ottenuta sostituendo il fattore α con il fattore β; l'applicazione di detta regola di produzione sarà denotata con $(x\,\alpha\,y \Rightarrow_G x\,\beta\,y)$

OSSERVAZIONI: le regole di derivazioni possono essere espresse sia mediante la forma classica:
in E.I. n°2 <unità> → minuti
<unità> → ore
oppure mediante la forma breve, ossia una forma abbreviata la quale ci consente di elencare un insieme di possibili simboli terminali all'interno della stessa riga, questo è possibile sono mediante l'uso dell'OR logico "|",
<unità> → minuti | ore

TRASFORMAZIONE, si ottiene partendo dall'assioma e applicando le regole di trasformazione sino ad ottenere solo simboli terminali, ogni trasformazione è dettata da una regola di produzione.

DERIVAZIONE IN UN PASSO, fissata una grammatica G, date due parole w, $z \in (\Sigma \cup M)^*$, una derivazione in un passo è l'applicazione di una regola di produzione che si indica con $w \Rightarrow_G z$ e si dice che: "z è derivabile in G da w in un passo".

DERIVAZIONE IN ZERO O PIÙ PASSI, si indica con "$\Rightarrow^*$ ",date due parole w e z, diremo $w \Rightarrow^* z$ se z può essere ottenuta da w applicando un numero finito di regole di produzione e si dice che "z è derivabile da w in zero o più passi di derivazione", indica l'applicazione di più regole, ossia una sequenza di derivazioni in un passo del tipo $w \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_m \Rightarrow z$. Queste si distinguono in due categorie:

- IN ZERO PASSI significa che w e z sono uguali, infatti: $w \Rightarrow_G^* z$, ma w = z quindi $w \Rightarrow_G^* w$
- IN PIÙ PASSI significa che devono esistere parole $x_1, x_2, \ldots, x_k$,t.c. $w \Rightarrow_G x_1 \Rightarrow_G x_2 \Rightarrow_G \ldots \Rightarrow_G x_k \Rightarrow_G z$ dove k indica il numero di passi di derivazione.

Uno dei metodi di rappresentazione di derivazione in più passi prende il nome di "rappresentazione ad albero", dove l'assioma viene posto in alto, centrale e a ogni livello inferiore si effettua una trasformazione, ponendo tra tag i metasimboli, che al livello successivo verranno ulteriormente trasformati, e fuori da tag i simboli terminali che al livello successivo non commuteranno il loro valore, i simboli terminali prendono altresì il nome di foglie dell'albero e costituiscono la parte che, se letta da sinistra verso destra, fornisce la trasformazione completa.

E.I. N°1: DERIVAZIONE IN PIÙ PASSI

<frase> $\Rightarrow_G^*$ il cane teme il gatto

partendo dall'assioma abbiamo operato più derivazioni fino ad ottenere solo simboli terminali:

<frase> $\Rightarrow_G$ <soggetto> <predicato> <complemento>
1° derivazione) <frase> $\Rightarrow_G$ <articolo> <nome> <predicato> <complemento>
2° derivazione) <frase> $\Rightarrow_G$ il <nome> <predicato> <complemento>
3° derivazione) <frase> $\Rightarrow_G$ il cane <predicato> <complemento>
4° derivazione) <frase> $\Rightarrow_G$ il cane teme <complemento>
5° derivazione) <frase> $\Rightarrow_G$ il cane teme <articolo><nome>
6° derivazione) <frase> $\Rightarrow_G$ il cane teme il <nome>
7° derivazione) <frase> $\Rightarrow_G$ il cane teme il gatto

si noti che nella derivazione in più passi le regole di produzione vengono applicate in ogni parola al primo metasimbolo da sinistra, il linguaggio generato è formato dalle parole $w \in$ {il, cane, gatto, topo, mangia, teme}* tali che <frase> $\Rightarrow_G^*$ w.

E.I. N°2: DERIVAZIONE IN PIÙ PASSI

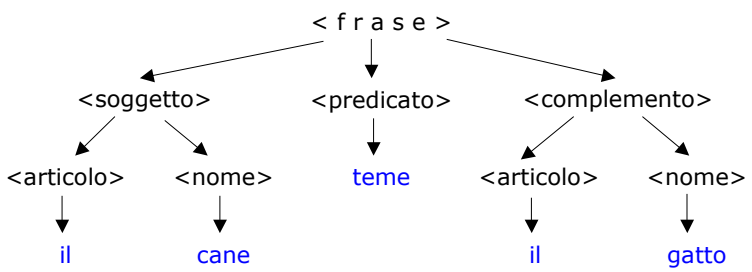
<messaggio> $\Rightarrow_G^*$ regionale da Milano viaggia con 2 ore di ritardo

partendo dall'assioma abbiamo operato più derivazioni fino ad ottenere solo simboli terminali:

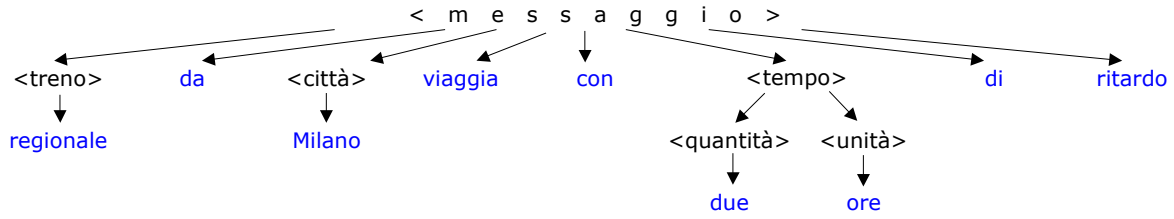
<messaggio> $\Rightarrow_G$ <treno> da <città> viaggia con <tempo> di ritardo
1° derivazione) <messaggio> $\Rightarrow_G$ regionale da <città> viaggia con <tempo> di ritardo
2° derivazione) < messaggio > $\Rightarrow_G$ regionale da Milano viaggia con <tempo> di ritardo
3° derivazione) < messaggio > $\Rightarrow_G$ regionale da Milano viaggia con <quantità><unità> di ritardo
4° derivazione) < messaggio > $\Rightarrow_G$ regionale da Milano viaggia con due <unità> di ritardo
5° derivazione) < messaggio > $\Rightarrow_G$ regionale da Milano viaggia con due ore di ritardo

si noti che nella derivazione in più passi le regole di produzione vengono applicate in ogni parola al primo metasimbolo, il linguaggio generato è formato dalle parole $w \in$ { regionale, interregionale, eurostar, Milano, Torino, Roma, Rimini, Foggia, Napoli, 1, 2, …, 60, minuti, ore}* tali che <messaggio> $\Rightarrow_G^*$ w.

E.I. N°1: RAPPRESENTAZIONE AD ALBERO



E.I. N°2: RAPPRESENTAZIONE AD ALBERO



IL LINGUAGGIO GENERATO DALLA GRAMMATICA G

$L_{(G)} = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

è un insieme di parole sull'alfabeto Σ, derivabili dall'assioma, ossia formate da simboli terminali; per ottenere simboli terminali si necessita di partire dall'assioma S ed effettuare le derivazioni in più passi, per indicare che la derivazione $\Rightarrow^*$ è stata effettuata sulla grammatica G si utilizza $\Rightarrow_G^*$.

F_I

F_I, è l'insieme delle forme sentenziali derivabili dall'assioma S in G in esattamente "i" passi di derivazione,

$F_i = \{ \delta \in (\Sigma \cup M)^* \mid \exists\, x_k \text{ con } 1 < k < i\text{-}1 \text{ t.c. } S \Rightarrow_G x_1 \Rightarrow_G x_2 \Rightarrow_G \dots \Rightarrow_G x_{i-1} \Rightarrow_G \delta \}$

T_I

T_I, è l'insieme delle parole del linguaggio Σ ottenute da S in i passi di derivazione, ossia un sottoinsieme di F_i, è quindi l'insieme dei simboli terminali ottenibili negli i passi di derivazione. $T_i = \{\delta \in F_i \mid \delta \in \Sigma^*\} = F_i \cap \Sigma^*$

Possiamo quindi fornire una seconda definizione di L(G) mediante l'utilizzo di Ti:

$$L_{(G)} = \bigcup_{i=0}^{\infty} T_i = T_1 \cup T_2 \cup T_3 \cup \dots \cup T_i \cup \dots \cup T_\infty = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$$

dove ricordiamo che: T_1 contiene tutte le parole ottenute con la derivazione in un passo, T_2 contiene tutte le parole ottenute con la derivazione in due passi, T_3 contiene tutte le parole ottenute con la derivazione in tre passi, …, T_k contiene tutte le parole ottenute con la derivazione in k passi.

GRAMMATICHE EQUIVALENTI, due grammatiche G_1 e G_2 si dicono equivalenti quando generano lo stesso linguaggio ossia quanto il linguaggio generato da G_1 ($L_{(G1)}$) è uguale al linguaggio generato da G_2 ($L_{(G2)}$),

$L_{(G1)} = L_{(G2)}$

ATTENZIONE equivalenti non significa aventi identica struttura e regole di derivazione.

PROCEDURA CHE ELENCA LE PAROLE GENERATE DALLA GRAMMATICA G

Necessitiamo ora di progettare una procedura che consente di generare sistematicamente tutte le parole derivanti dall'assioma, sul quale vengono applicate tutte le possibili combinazioni di regole di produzione.

```
Procedura Elenca ()
{
        F0 = {S}                    # derivando in zero passi(F0)  l'assioma (S) ottengo l'assioma stesso
        i = 1
        while (i > 0) do
                costruisci Fi
                Ti = Fi ∩ Σ*
                Output(Ti)
                i = i+1
}
```

ANALISI DESCRITTIVA DELLA PROCEDURA ELENCA:

- Non ha alcun input;
- Deriva al passo zero l'assioma S, dato che effettuare una derivazione in zero passi significa non ottenere alcuna trasformazione, la derivazione in zero passi fornisce l'assioma stesso;
- Inizializza la variabile "i" ad uno, questa variabile sarà il contatore per gli i passi di derivazione;
- effettua un ciclo di while il quale, finché i > 0, effettua la derivazione al passo i-esimo.
- Crea allo stesso tempo T al passo i-esimo
- Effettua la stampa (output) di T_i, ossia stampa l'insieme delle parole ottenute dalla derivazione del passo i
- Incrementa la variabile contatore passi per riprendere il ciclo while

Si tratta ovviamente di una procedura che non termina mai, effettua una serie di derivazioni in i passi e stampa le parole derivanti da suddette derivazioni ma non comporta alcun tipo di terminazione della procedura stessa.

L RICORSIVAMENTE NUMERABILE ⇔ L AMMETTE G CHE LO GENERA

Per poter dimostrare che $\exists$ G | $L_{(G)}$ = L necessitiamo di dimostrare due punti distinti:

A. Se L ammette una grammatica G ⇒ L è enumerabile

B. Se L è enumerabile ⇒ ammette una grammatica G

in questo corso ci poniamo il problema di dimostrare solo il punto A. in quanto il punto B. richiede delle nozioni non in nostro possesso e che non vengono fornite in maniera esauriente nel corso.

A. SE L AMMETTE UNA GRAMMATICA G ⇒ L È ENUMERABILE

Per procedere con la dimostrazione dobbiamo esibire una procedura "w" che sulle parole del linguaggio $L_{(G)}$ restituisca 1, mentre sulle parole non appartenenti a $L_{(G)}$ non termini. Per effettuare detta dimostrazione so di essere in possesso di una grammatica G, inoltre sono in possesso della procedura elenca sopra enunciata, ora non devo far altro che trasformare "elenca" in "w".

TRASFORMAZIONE DI "ELENCA" IN "W":

per trasformare "elenca" in "w" devo fornirgli dei parametri.

1) Passiamo ad "elenca" una parola "x" in Σ^*;
2) se x è una parola che appartiene al linguaggio generato dalla grammatica G allora esiste una derivazione per "x", ossia: $S \Rightarrow_G x_1 \Rightarrow_G x_2 \Rightarrow_G \dots \Rightarrow_G x_k \Rightarrow_G x$ questo significa che prima o poi la parola sarà contenuta nell'insieme delle parole del linguaggio ($x \in T_i$). Quando $T_i = x$ allora la procedura elenca proverà a fermarsi e restituire 1, in questo modo ho risolto il problema di trovare x. Se $x \notin T_i$ devo costruire il T_i del passo successivo per verificare se x appartiene al prossimo passo di derivazione in più passi, continuo a derivare fino ad ottenere $x \in T_i$. Questa seconda modifica quindi consiste nel sostituire l'istruzione "output(T_i)" con un istruzione in grado di verificare se x appartiene al passo di derivazione attualmente analizzato.

Operiamo ora le modifiche su "elenca" al fine di trasformarla in "w"

```
Procedura Elenca ()
    F0 = {S}
    i = 1
    while (i > 0) do
        costruisci Fi
        Ti = Fi ∩ Σ*
        Output(Ti)
        i = i+1
```

```
Procedura w (x ∈ Σ*)
    F0 = {S}
    i = 1
    while (i > 0) do
        costruisci Fi
        Ti = Fi ∩ Σ*
        If (x ∈ Ti) return '1'
        i = i+1
```

testiamo ora la correttezza della procedura "w"

CASO I: TEST DI W PASSANDO IN INPUT $x \in L_{(G)}$

Passiamo alla procedura "w" la parola $x \in L_{(G)}$.Se $x \in L_{(G)}$ viene implicata l'esistenza di una derivazione in G per la parola x, ossia: $S \Rightarrow_G x_1 \Rightarrow_G x_2 \Rightarrow_G \dots \Rightarrow_G x_k \Rightarrow_G x$

Dato che abbiamo una generazione allora abbiamo un certo numero di passi, (che indicheremo per convenzione pari a k+1) in cui la parola x viene generata per derivazione in G da S. Per definizione di T_i, se x viene generata in k+1 passi di derivazione, allora $x \in T_{k+1}$,dove T_{k+1} contiene le parole del linguaggio ottenute da S in k+1 passi di derivazione. Il ciclo while della procedura quando i = k+1 costruirà F_{k+1} e creerà T_{k+1}, ritornando "1" non appena verrà eseguita l'istruzione if.

CASO II: TEST DI W PASSANDO IN INPUT $x \notin L_{(G)}$

Passiamo alla procedura "w" la parola $x \notin L_{(G)}$.Se $x \notin L_{(G)}$ viene implicata la non esistenza di alcuna derivazione in G in grado di generare la parola x, ossia: non esiste T_i contenente x.

Dato che per definizione $L_{(G)} = \cup T_i$, e dato che non esiste T_i in grado di generare x, si ottiene che la procedura "w" non termina, continuando a derivare in i passi senza ottenere mai la parola x.

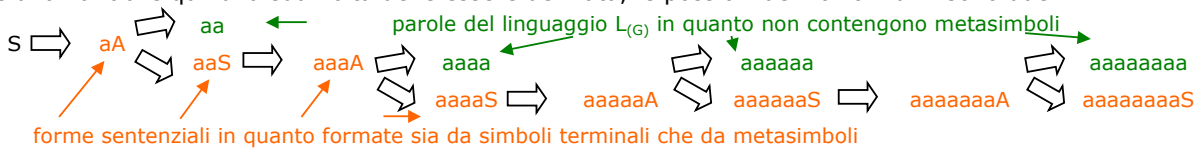
ESEMPIO DI GRAMMATICA

$G = < \Sigma = \{a\}, M = \{S, A\}, P = \{S \rightarrow aA, A \rightarrow aS, A \rightarrow a\}, S>$

Partiamo dall'assioma, ossia da S ed effettuiamo le possibili derivazioni:

$S \Rightarrow_G aA$

A è una variabile quindi a sua volta deve essere derivata, le possibili derivazioni di A sono due:



questa grammatica è una grammatica infinita che produce solo parole {a}* con esponente pari, ossia $L_{(G)} = \{a^{2n} \mid n \geq 1\}$

ESEMPIO DAL LINGUAGGIO ALLA GRAMMATICA

sia L il linguaggio formato dalle espressioni aritmetiche parentesizzate, ossia che sfruttano l'utilizzo delle parentesi (,); ottenute a partire dalle variabili x, y e dalle costanti 0, 1, sulle operazioni +, *; dove esempio di parole di L sono:

x, 1, (x + y), (y + (x * 1)), (((0 * x) + 1) + (y * x)), …

dato "espressione" quale assioma, possiamo quindi descrivere il linguaggio L nel seguente modo :

<espressione> → x

<espressione> → 1

<espressione> → (x + y)

<espressione> → (1 + (1 * y))

<espressione> → (((0 * 1) + y) + (1 * x))

…

sono infinite quindi devo trovare una regola che mi permetta di definirle tutte, costruiamo quindi una grammatica G = <Σ, M, P, S>, in grado di generare L:

- alfabeto terminale Σ = { x, y,0, 1, +, *, (,) }
- alfabeto non terminale M = {<espressione>, <variabile>, <costante>, <operazione>}
- P = { 1) <espressione> → <costante>
  2) <espressione> → <variabile>
  3) <espressione> → (<espressione> <operazione> <espressione>)
  4) <costante> → 0 | 1
  5) <variabile> → x | y
  6) <operazione> → + | * }
- Assioma S = <espressione>

G = < Σ={x, y,0, 1, +, *, (,) } , M={<espressione>, <variabile>, <costante>, <operazione>} ,
P={ <espressione> → <costante> | <variabile> | <espressione> <operazione> <espressione>
<costante> → 0 | 1, <variabile> → x | y, <operazione> → + | * } , <espressione> >

Proviamo ad ottenere: (0 + (x * 1)) mediante la derivazione in più passi:

1° derivazione) <espressione> $\Rightarrow_G^{3)}$ (<espressione> <operazione> <espressione>)

2° derivazione) <espressione> $\Rightarrow_G^{1)}$ (<costante> <operazione> <espressione>)

3° derivazione) <espressione> $\Rightarrow_G^{4)}$ (0 <operazione> <espressione>)

4° derivazione) <espressione> $\Rightarrow_G^{6)}$ (0 + <espressione>)

5° derivazione) <espressione> $\Rightarrow_G^{3)}$ (0 + (<espressione> <operazione> <espressione>))

6° derivazione) <espressione> $\Rightarrow_G^{2)}$ (0 + (<variabile> <operazione> <espressione>))

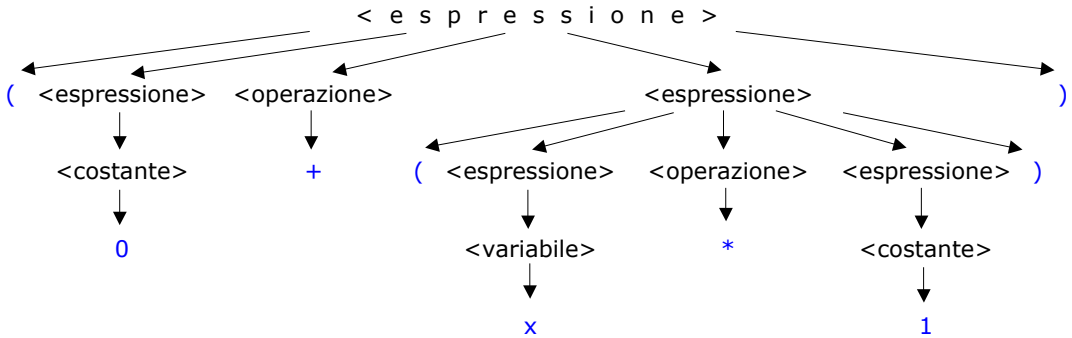
7° derivazione) <espressione> $\Rightarrow_G^{5)}$ (0 + (x <operazione> <espressione>))

8° derivazione) <espressione> $\Rightarrow_G^{6)}$ (0 + (x * <espressione>))

9° derivazione) <espressione> $\Rightarrow_G^{1)}$ (0 + (x * <costante>))

10° derivazione) <espressione> $\Rightarrow_G^{4)}$ (0 + (x * 1))

Proviamo ad ottenere: (0 + (x * 1)) mediante la rappresentazione ad albero:



ESERCIZIO

Dato il linguaggio L = linguaggio delle parole formate sulle parentesi "(", ")" t.c. ogni parola w = (z) per un'opportuna scelta di z.

Regole di composizione:

- ogni parentesi aperta deve essere chiusa
- non si può chiudere una parentesi precedentemente non aperta

date le seguenti parole definire quali non appartengono al linguaggio appena enunciato:

1) ( ) w = (z) z = ε
2) (( )) w = (z) z = ()
3) ((( ) ( ))) w = ((zz)) z = ()
4) ( ) ( ) non è una parola ben formata in quanto prima chiude e poi apre una parentesi

strutturiamo in maniera ufficiale le regole:

A) $S \rightarrow ()$
B) $S \rightarrow (S)$
C) $S \rightarrow (SS)$

TEST: Costruiamo con le regole appena definite la seguente parola del linguaggio: ((( ) ( )))

S $\Rightarrow^A$ (S)
(S) $\Rightarrow^B$ ((SS))
((S S)) $\Rightarrow^A$ ((( ) S))
((( ) S)) $\Rightarrow^A$ ((( ) ( )))

TEST: costruiamo sempre con le regole appena definite la seguente parola del linguaggio: (( ) ( ) ( ))
la suddetta parola non è generabile con le regole di produzione appena proposte, si necessiterebbe della regola: $S \rightarrow (SSS)$ in sostanza queste regole mi generano solo una parte del linguaggio sopra indicato.

ESERCIZIO

Sia L il linguaggio delle parole formate sulle parentesi (,) t.c. w = (z) per un'opportuna scelta di z.

Regole di composizione:

- ogni parentesi aperta deve essere chiusa
- non si può chiudere una parentesi precedentemente non aperta

costruiamo quindi una grammatica G = <Σ, M, P, S>, in grado di generare L:

- alfabeto terminale Σ = { (,) }
- alfabeto non terminale M = {S, X}
- P = { 1) $S \rightarrow (X$
  2) $X \rightarrow)$
  3) $X \rightarrow (XX$ }
- Assioma S

G = < Σ={ (,) } , M={S, X} , P={$S \rightarrow (X$, $X \rightarrow)$ | (XX} , S >

F_0 = {S} F_1 = {(X} F_2 = {(), ((XX} F_3 = {(()X, (((XXX, ((X), ((X(XX}
T_0 = ∅ T_1 = ∅ T_2 = {()} T_3 = ∅
F_4 = {((()), (()(XX, (((()XX, (((X)X, (((XX), ((((XXXX, (((X(XXX, (((XX(XX, (()), (((XX), (()(XX, ((X()X, ((X(X), (((XX(XX, ((X((XX, ((X(X(XX}
T_4 = {(())}
...

TEST: Costruiamo con la grammatica appena definita la seguente parola del linguaggio L: ((( ) ( )))

S $\Rightarrow^1$ (X
(X $\Rightarrow^3$ ((XX
((XX $\Rightarrow^3$ (((XXX
(((XXX $\Rightarrow^2$ ((( ) XX
((( ) XX $\Rightarrow^3$ ((( ) (XXX
((( ) XX $\Rightarrow^{2*}$ ((( ) ( )))

TEST: Costruiamo con la grammatica appena definita la seguente parola del linguaggio L: (( ) ( ) ( ))

S $\Rightarrow^1$ (X $\Rightarrow^3$ ((XX $\Rightarrow^2$ (( ) X $\Rightarrow^3$ (( ) (XX $\Rightarrow^2$ (( ) ( ) X $\Rightarrow^3$ (( ) ( ) (XX $\Rightarrow^*$ (( ) ( ) ( ))

I DOCUMENTI XML

Cos'è un documento XML? Un documento XML è un documento contenente testo e marcatori (anche chiamati tag), questi ultimi forniscono informazioni aggiuntive rispetto al testo che fornisce solo informazioni descrittive.

<mail>
<mittente> Francesco </mittente>
<destinatario> Edoardo </destinatario>
<corpo> ciao come va? Tutto bene </corpo>
</mail>

testo: Francesco, Edoardo, ciao come va? Tutto bene

Marcatori: mail, mittente, destinatario, corpo i quali concorrono a fornire informazioni sul testo, il marcatore mail fornisce l'informazione aggiuntiva sul fatto che non si tratta di un qualsiasi testo XML ma di un testo da utilizzare per un e-mail, mittente mi fornisce l'informazione aggiuntiva sul fatto che "Francesco" oltre ad essere parte del testo è altresì il nome del mittente dell'e-mail

ESERCIZIO SUI DOCUMENTI XML

Consideriamo l'alfabeto dei tag di apertura e chiusura indicando con "!a" il tag di chiusura e con "a" il tag di apertura, otteniamo quindi il seguente alfabeto:

$\Sigma = \{a, !a, b, !b\}$

le regole della grammatica dei tag sono:

- Ogni qualvolta venga chiuso un tag, si necessita di verificare che precedentemente il tag fosse stato aperto
- Ogni qualvolta venga chiuso un tag, si necessita di verificare che tutti i tag aperti successivamente (all'interno del tag raccoglitore) siano stati chiusi
- Deve esistere un tag che contenga tutto il resto

Date le regole appena definite, indicare quali delle seguenti parole appartengono al linguaggio XML:

a b !b !a $\in$ XML ? SI

a b !a !b $\in$ XML ? no, in quanto il tag "a" viene chiuso prima della chiusura del tag che "a" contiene, ossia prima della chiusura di "b"

a !a b !b $\in$ XML? No, in quanto non esiste un tag che contiene tutto il resto

a b !b a !a $\in$ XML? No, in quanto il tag "a" è stato aperto due volte ma chiuso solo una volta

a b !b a !a !a $\in$ XML? SI

Costruiamo ora la grammatica $G_{XML} = <\Sigma, M, P, S>$, in grado di generare L_{XML}:

- Alfabeto terminale $\Sigma = \{ a, !a, b, !b \}$
- Alfabeto non terminale $M = \{S, A, B\}$
- P = { 1) $S \rightarrow aA$ 2) $A \rightarrow !a$
  3) $A \rightarrow aAA$ 4) $A \rightarrow bBA$
  5) $B \rightarrow !b$ 6) $B \rightarrow bBB$
  7) $B \rightarrow aAB$
- Assioma S

$G = < \Sigma=\{a, !a, b, !b\} , M=\{S, A, B\} , P=\{S \rightarrow aA , A \rightarrow !a \mid aAA \mid bBA , B \rightarrow !b \mid bBB \mid aAB\} , S >$

TEST: Costruiamo con la grammatica appena definita la seguente parola del linguaggio L_{XML}: a b !b !a

$S \Rightarrow^1 a\ A \Rightarrow^4 a\ b\ B\ A \Rightarrow^5 a\ b\ !b\ A \Rightarrow^2 a\ b\ !b\ !a$

TEST: Costruiamo la parola di L_{XML}: a b b !b a !a !b !a

$S \Rightarrow^1 a\ A \Rightarrow^4 a\ b\ B\ A \Rightarrow^4 a\ b\ b\ B\ B\ A \Rightarrow^5 a\ b\ b\ !b\ B\ A \Rightarrow^7 a\ b\ b\ !b\ a\ A\ B\ A \Rightarrow^2 a\ b\ b\ !b\ a\ !a\ B\ A \Rightarrow^5$
$\Rightarrow^5 a\ b\ b\ !b\ a\ !a\ !b\ A \Rightarrow^2 a\ b\ b\ !b\ a\ !a\ !b\ !a$

ESERCIZIO

Sia L il linguaggio = $\{a^n b^n \mid n \geq 1\}$

costruiamo ora la grammatica $G_L = <\Sigma, M, P, S>$, in grado di generare L:

- alfabeto terminale $\Sigma = \{ a, b \}$
- alfabeto non terminale $M = \{S\}$
- P = { 1) $S \rightarrow aSb$
  2) $S \rightarrow ab$
- Assioma S

$G = < \Sigma=\{a, b \} , M=\{S \} , P=\{S \rightarrow aSb \mid ab\} , S >$

$F_0 = \{S\}$ $T_0 = \varnothing$

$F_1 = \{ab, aSb\}$ $T_1 = \{ab\}$

$F_2 = \{aabb, aaSbb\}$ $T_2 = \{aabb\}$

...

TEST: Costruiamo con la grammatica appena definita la seguente parola del linguaggio L: a^2b^2

$S \Rightarrow^1 aSb \Rightarrow^2 aabb$

TEST: Costruiamo con la grammatica appena definita la seguente parola del linguaggio L: a^4b^4

$S \Rightarrow^1 aSb \Rightarrow^1 aaSbb \Rightarrow^1 aaaSbbb \Rightarrow^2 aaaabbbb$

ESERCIZIO

Sia L il linguaggio = $\{a^n b^n c^n \mid n \geq 1\}$

costruiamo ora la grammatica $G_L = <\Sigma, M, P, S>$, in grado di generare L:

- alfabeto terminale $\Sigma = \{ a, b, c \}$
- alfabeto non terminale M = {S, B, C}
- P = { 1) $S \rightarrow aSBC$
  3) $S \rightarrow aBC$
  4) $CB \rightarrow BC$
  5) $aB \rightarrow ab$
  6) $bB \rightarrow bb$
  7) $bC \rightarrow bc$
  8) $cC \rightarrow cc$
- Assioma S

G = < Σ={a, b, c} , M={S, B, C} , P={$S \rightarrow aSBC|aBC$, $CB \rightarrow BC$, $aB \rightarrow ab$, $bB \rightarrow bb$, $bC \rightarrow bc$, $cC \rightarrow cc$} , S >

Vediamo in dettaglio ciascuna regola di produzione "P" cosa genera:

1) $S \rightarrow aSBC$ # cosa genera? $S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaSBCBCBC$, il nostro linguaggio genera parole del tipo $a^n b^n c^n$, ossia, in quest'esempio appena proposto dovrebbe creare la parola $a^3b^3c^3$, ossia la parola "aaabbbccc" che invece non è stata creata. Che cosa bisogna fare? Si necessita dell'utilizzo della regola di produzione 3)
2) $S \rightarrow aBC$
3) $CB \rightarrow BC$ # questa regola mi consente di invertire il metasimbolo C con il metasimbolo B, è una regola di produzione in grado di ottenere una serie di permutazioni che mi permettano di ordinare i simboli di cui la parola si compone nell'ordine bbb ... ccc.
4) $aB \rightarrow ab$
5) $bB \rightarrow bb$
6) $bC \rightarrow bc$
7) $cC \rightarrow cc$

4), 5), 6) e 7) sono regole dipendenti dal contesto, ossia regole di produzione che non consentono il loro utilizzo se non quando il testo si trova nello stato di pre-trasformazione ordinata, ciascuna di queste regole quindi mi consente di trasformare un metasimbolo in un simbolo terminale solo quando detto metasimbolo è preceduto da un altro simbolo terminale.

TEST: costruzione, con la grammatica appena definita, della seguente parola del linguaggio L: $a^2b^2c^2$

$S \Rightarrow^1 aSBC \Rightarrow^2 aaBCBC \Rightarrow^3 aaBBCC \Rightarrow^4 aabBCC \Rightarrow^5 aabbCC \Rightarrow^6 aabbcC \Rightarrow^7 aabbcc$

TEST: Costruiamo con la grammatica appena definita la seguente parola del linguaggio L: $a^3b^3c^3$

$S \Rightarrow^1 aSBC \Rightarrow^1 aaSBCBC \Rightarrow^2 aaaBCBCBC \Rightarrow^3 aaaBBCCBC \Rightarrow^3 aaaBBCBCC \Rightarrow^3 aaaBBBCCC \Rightarrow^4 aaabBBCCC \Rightarrow^5 aaabbBCCC \Rightarrow^5 aaabbbCCC \Rightarrow^6 aaabbbcCC \Rightarrow^7 aaabbbccC \Rightarrow^7 aaabbbccc$

La parola $a^3b^3c^3$ è derivabile da S in G solo mediante l'esecuzione di questa precisa combinazione di sequenza di regole di produzione: ⇒ 1, 1, 2, 3, 3, 3, 4, 5, 5, 6, 7, 7, ricordiamo che le regole devono essere applicate in tutte le combinazioni possibili, il risultato voluto invece, è ottenuto solo mediante l'applicazione della giusta sequenza di regole, vediamo se si ottiene la parola $a^3b^3c^3$ anche mediante l'applicazione di una diversa sequenza di regole di produzione:

$S \Rightarrow^1 aSBC \Rightarrow^1 aaSBCBC \Rightarrow^2 aaaBCBCBC \Rightarrow^3 aaaBBCCBC \Rightarrow^4 aaabBCCBC \Rightarrow^5 aaabbCCBC \Rightarrow^6 aaabbcCBC \Rightarrow^7 aaabbccBC \Rightarrow^?$ Non sono applicabili ulteriori regole di produzione, quindi non riesco, mediante l'applicazione di questa sequenza di regole di produzione, ad ottenere alcun tipo di parola di L.

GERARCHIA DI CHOMSKY

È possibile classificare le grammatiche in funzione del tipo di regola di produzione. Un'interessante classificazione è quella proposta da Chomsky in base a considerazioni di carattere linguistico e di semplicità di trattazione matematica. E' considerata una gerarchia decrescente di quattro tipi di grammatiche:

- grammatica di tipo 0, le regole di produzione sono arbitrarie;
- grammatica di tipo 1, ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che $|\beta| \geq |\alpha|$, ossia non vengono consentite cancellazioni durante le trasformazioni;
- grammatica di tipo 2, ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che α sia un metasimbolo;
- grammatica di tipo 3, ogni regola di produzione $\alpha \rightarrow \beta$ deve essere tale che si presenti in una delle seguenti due forme: $\alpha \rightarrow x$ oppure $\alpha \rightarrow y\beta$, con α e β metasimboli e x e y simboli terminali;

RICORDANDO PRIMA CHE: si parla di regola di produzione quando si ha una regola della forma $\alpha \rightarrow \beta$:

GRAMMATICA DI TIPO 0: VERSIONE UNICA

Si parla di grammatica di tipo 0 quando le regole di produzione sono arbitrarie, ossia, quando non esiste alcun tipo di restrizione su dette regole.

- $\alpha \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
- $\beta \in (\Sigma \cup M)^*$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M inclusa ε

Appartengono alle grammatiche di tipo 0 qualsiasi grammatica costruibile mediante regole di produzione.

GRAMMATICA DI TIPO 1: I VERSIONE

una grammatica si dice di tipo 1 e prende il nome di grammatica contestuale se ogni regola di produzione $\alpha \rightarrow \beta$ soddisfa le seguenti condizioni:

- $\alpha \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
- $\beta \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
- $|\beta| \geq |\alpha|$

Si parla quindi di grammatica di tipo 1 se, in ogni regola di produzione $\alpha \rightarrow \beta$ della grammatica, viene soddisfatta la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$.
Si possono definire regole che trasformano o incrementano i simboli di α per ottenere β, ma non si possono definire regole in cui β è ottenuto mediante la cancellazione di simboli di α. Non sono quindi ammesse regole della forma $\alpha \rightarrow \varepsilon$, questo è giustificato dal fatto che la lunghezza della parola vuota è zero ($|\varepsilon| = 0$) e quindi inferiore alla lunghezza di qualsiasi α utilizzato nella regola di produzione per ottenere ε, infatti, $\forall\ \alpha \in (\Sigma \cup M)^+$, $|\varepsilon| < |\alpha|$. Non sono altresì ammesse regole che consentono cancellazioni di simboli durante una trasformazione.
Le grammatiche di tipo 1 prendono il nome di grammatiche a lunghezza non decrescente, sarebbe errato chiamarle grammatiche a lunghezza crescente giacché non viene garantita una crescita della lunghezza delle parole nei diversi passi di derivazione, viene invece garantita una non decrescita.
Poiché ogni grammatica contestuale è chiaramente di tipo 1 ed è possibile provare che per ogni grammatica G di tipo 1 esiste una grammatica G' contestuale che genera lo stesso linguaggio, i linguaggi di tipo 1 sono esattamente i linguaggi dipendenti dal contesto.

GRAMMATICHE DI TIPO 2: I VERSIONE

una grammatica si dice di tipo 2 se ogni regola di produzione $\alpha \rightarrow \beta$ soddisfa le seguenti condizioni:

- $\alpha \in M$ è l'insieme dei metasimboli,
- $\beta \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,

Si parla di grammatica di tipo 2 se, in ogni regola di produzione $\alpha \rightarrow \beta$ della grammatica, viene soddisfatta la condizione che α è un metasimbolo, ossia $\alpha \in M$, β è una forma sentenziale, ossia una sequenza di simboli e/o metasimboli, ad eccezione della parola vuota, che come nelle grammatiche di tipo 1, viene esclusa dalle derivazioni, ossia $\beta \in (\Sigma \cup M)^+$.

GRAMMATICHE DI TIPO 3: I VERSIONE

una grammatica si dice di tipo 3 se ogni regola di produzione assume le seguenti due forme:

- $\alpha \rightarrow x$
  - $\alpha \in M$
  - $x \in \Sigma^+$
- $\alpha \rightarrow y\beta$
  - $\alpha \in M$
  - $y \in \Sigma^*$
  - $\beta \in M$

Si parla di grammatica di tipo 3 se, in ogni regola di produzione della grammatica, assume le seguenti due forme:

- $\alpha \rightarrow x$ con $\alpha \in M$, $x \in \Sigma^+$

viene soddisfatta la condizione che α è un metasimbolo, ossia $\alpha \in M$, x è una sequenza di simboli terminali esclusa la parola vuota ε, ossia una sequenza di simboli non contenenti metasimboli, $x \in \Sigma^+$

- $\alpha \rightarrow y\beta$ con $\alpha \in M$, $y \in \Sigma^*$ e $\beta \in M$

viene soddisfatta la condizione che α è un metasimbolo, ossia $\alpha \in M$, y è una sequenza di simboli terminali, ossia una sequenza di simboli non contenenti metasimboli, ad eccezione della parola vuota (questo contrasta con le grammatiche di tipo 2, le quali non consentivano a β di assumere valore ε, invece nelle grammatiche di tipo 3 y può anche assumere valore ε), ossia $y \in \Sigma^*$ e β è un metasimbolo, ossia $\beta \in M$.

ESERCIZIO

Sia G la grammatica per il linguaggio $L = a^{2n}$ con $n \geq 1$

$G = < \Sigma=\{a\} , M=\{S\} , P=\{S \rightarrow aaS \mid aa\} , S >$

Determinare il tipo di grammatica di G

- G è di tipo 0?
  SI in quanto è una grammatica improntata su regole di produzione
- G è di tipo 1?
  SI in quanto è una grammatica che soddisfa la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$, le regole di produzione di G, infatti, non consentono di ottenere β mediante la cancellazione di simboli di α.
- G è di tipo 2?
  SI in quanto è una grammatica che soddisfa la condizione che α è un metasimbolo, ($\alpha \in M$), β è una forma sentenziale, ossia una sequenza di simboli e/o metasimboli, ad eccezione della parola vuota, ($\beta \in (\Sigma \cup M)^+$)
- G è di tipo 3?
  SI, infatti, ogni regola di produzione di G, assume solo le due forme consentite per il tipo 3:
  - $S \rightarrow aaS$ è della forma $\alpha \rightarrow y\beta$ con $\alpha=S \in M$, $y=$"aa" $\in \Sigma^*$ e $\beta=S \in M$
  - $S \rightarrow aa$ è della forma $\alpha \rightarrow x$ con $\alpha=S \in M$, $x=$"aa" $\in \Sigma^+$

G è una grammatica di tipo 3.

ESERCIZIO

Sia G la grammatica per il linguaggio $L = a^n b^n$ con $n \geq 1$

$G = < \Sigma=\{a, b\} , M=\{S\} , P=\{S \rightarrow aSb \mid ab\} , S >$

Determinare il tipo della grammatica G

- G è di tipo 0?
  SI, infatti, è una grammatica improntata su regole di produzione
- G è di tipo 1?
  SI, infatti, è una grammatica che soddisfa la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$, le regole di produzione di G, infatti, non consentono di ottenere β mediante la cancellazione di simboli di α.
- G è di tipo 2?
  SI in quanto è una grammatica che soddisfa la condizione che α è un metasimbolo, ($\alpha \in M$), β è una forma sentenziale, ossia una sequenza di simboli e/o metasimboli, ad eccezione della parola vuota, ($\beta \in (\Sigma \cup M)^+$)
- G è di tipo 3?
  NO, non vengono infatti soddisfatte le due forme previste per le grammatiche di tipo 3:
  - $S \rightarrow ab$ è della forma $\alpha \rightarrow x$ con $\alpha=S \in M$, $x=$"ab" $\in \Sigma^+$
  - $S \rightarrow aSb$ non è né della forma $\alpha \rightarrow x$, né della forma $\alpha \rightarrow y\beta$

G è quindi una grammatica di tipo 2

ESERCIZIO

Sia G la grammatica per il linguaggio $L = a^n b^n c^n$ con $n \geq 1$

$G = < \Sigma=\{a, b, c\} , M=\{S, B, C\} , P=\{S \rightarrow aSBC|aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\} , S >$

Determinare il tipo di grammatica a cui appartiene G

- G è di tipo 0?
  SI in quanto è una grammatica improntata su regole di produzione
- G è di tipo 1?
  SI in quanto è una grammatica che soddisfa la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$, le regole di produzione di G, infatti, non consentono di ottenere β mediante la cancellazione di simboli di α.
- G è di tipo 2?
  NO in quanto non viene soddisfatta la condizione su $\alpha \in M$:
  - $CB \rightarrow BC$ non soddisfa la condizione che persista solo UN metasimbolo per α, $\alpha \notin M$ bensì $\alpha \in M^*$
  - $aB \rightarrow ab$ non soddisfa la condizione che α sia metasimbolo, $\alpha \notin M$ bensì $\alpha \in (M \cup \Sigma)^*$

G è quindi una grammatica di tipo 1

CLASSE DEI LINGUAGGI

Ogni grammatica G genera un linguaggio $L_{(G)}$. una classificazione delle grammatiche porta ad una naturale classificazione dei linguaggi

Un linguaggio si definisce di tipo k (con k che può assumere i valori 0, 1, 2 e 3) quando ammette una grammatica G di tipo k che lo genera, ossia quando $L = L_{(G)}$.

Indicheremo con R_k la classe dei linguaggi di tipo k, esistono quindi 4 classi di linguaggi:

- classe R_3 che prende anche il nome di classe dei linguaggi regolari
- classe R_2 che prende anche il nome di classe dei linguaggi acontenstuali, o anche detti linguaggi liberi da contesto in quanto sono linguaggi che non hanno alcuna dipendenza dal contesto

  ESEMPIO DI REGOLA LIBERA DA CONTESTO (NON CONTESTUALE)

  $B \rightarrow b$ prende il nome di regola libera da contesto in quanto può essere applicata senza restrizioni alcune o vincoli temporali
- classe R_1 che prende anche il nome di classe dei linguaggi contestuali, o anche detti linguaggi dipendenti dal contesto

  ESEMPIO DI REGOLA DIPENDENTE DA CONTESTO

  $aB \rightarrow ab$ prende il nome di regola dipendente da contesto in quanto può essere applicata, nell'esempio specifico, solo qual'ora il metasimbolo "B" sia preceduto dal simbolo terminale "a", in caso contrario la regola non può essere applicata, non è applicabile infatti per esempio nei seguenti casi: "aa<u>bB</u>aaC", "aa<u>AB</u>aab", "a<u>bB</u>", … in quanto nessuno di questi casi appartiene al contesto richiesto dalla regola di produzione definita.
- classe R_0 non hanno un nome specifica ma rientrano nella definizione di linguaggi ricorsivamente numerabili in quanto definiti dall'essere costruibili da una grammatica

ESEMPIO DI CLASSIFICAZIONE DI LINGUAGGI

sia $L = a^n b^n c^n$ con $n \geq 1$, L ammette una grammatica G di tipo 1 che lo genera, ossia $\exists$ G tipo 1 | $L = L_{(G)}$, il linguaggio L è quindi un linguaggio di tipo 1.

ESEMPIO DI CLASSIFICAZIONE DI LINGUAGGI

sia $L = a^n b^n$ con $n \geq 1$, L ammette una grammatica G di tipo 2 che lo genera, ossia $\exists$ G tipo 2 | $L = L_{(G)}$, il linguaggio L è quindi un linguaggio di tipo 2.

ESEMPIO DI CLASSIFICAZIONE DI LINGUAGGI

sia $L = a^{2n}$ con $n \geq 1$, L ammette una grammatica G di tipo 3 che lo genera, ossia $\exists$ G tipo 3 | $L = L_{(G)}$, il linguaggio L è quindi un linguaggio di tipo 3.

IL GRAFO (GR(X))

Un grafo è una rappresentazione grafica che consente di visualizzare tutte le possibili trasformazioni operabili all'interno di una grammatica.

Il grafo consta di due elementi: $GR(x) = (V_x, E_x)$

V_x sono i vertici del grafo, ossia l'insieme delle forme sentenziali "y" costruibili nella grammatica G;

E_x sono gli archi del grafo, ossia l'insieme delle coppie (y, y') tali per cui, nella grammatica G, esista una derivazione in un passo in grado di trasformare y in y', ovvero $\exists$ d | $y \Rightarrow^d y'$

effettuare la costruzione del grafo GR(x) consiste nell'inserimento, in colonne, delle forme sentenziali di cui la grammatica si compone, dette forme sentenziali costituiranno i vertici del grafo (Vx). Data quindi una grammatica $G = < \Sigma=\{x\} , M=\{S\} , P=\{…\} , S >$, si necessita la costruzione dei vertici del grafo, come si inseriscono nel grafo?

Nella colonna 1 vengono inserite le forme sentenziali aventi lunghezza 1 (x, S);

Nella colonna 2 vengono inserite le forme sentenziali aventi lunghezza 2 (xx, xS, Sx, SS);

Nella colonna 3 vengono inserite le forme sentenziali aventi lunghezza 3 (xxx, xxS, xSx, xSS, …);

…

ora si necessita della costruzione di quei collegamenti che permettono di visualizzare le relazioni tra i vertici. Gli archi sono costruibili seguendo le regole di produzioni contenute nell'insieme P della grammatica.

Qualora si necessitasse di stabilire se una parola $w \in \Sigma^*$, dove Σ è l'alfabeto della grammatica G, appartiene al linguaggio generato da G (L(G)), dato il grafo GR(x), sarà sufficiente stabilire se, partendo da S e seguendo gli archi del grafo, sia possibile raggiungere la parola cercata (w): (S, y_1), (y_1, y_2), $(y_2, y_…)$, $(y_…, y_k)$, (y_k, w)

ossia si verifica la seguente situazione in GR(x) :

TEOREMA DELL'INCLUSIONE

RICORDANDO PRIMA CHE: si dice che l'insieme A è incluso (è contenuto) in un insieme B, $A \subseteq B$, se:

- $\forall a \in A, a \in B$ # ogni elemento di A è anche elemento di B
- A può essere uguale a B

RICORDANDO PRIMA CHE: si dice che l'insieme A è incluso strettamente (o propriamente), $A \subset B$, se:

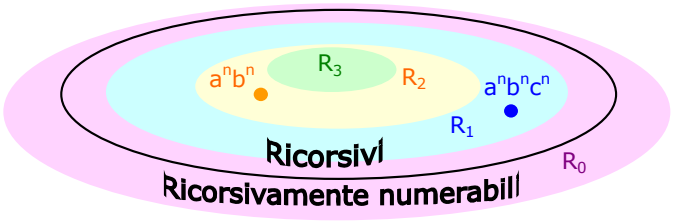
- $\forall a \in A, a \in B$ # ogni elemento di A è anche elemento di B
- $\exists$ almeno un $b \in B \mid b \notin A$ # B contiene almeno un elemento non appartenente ad A

Chiaramente le grammatiche di tipo k sono sottoinsieme proprio delle grammatiche di tipo j, se $k < j$ questo implica la seguente relazione di inclusione per le classi R_k:

R_0 è la classe dei linguaggi più ampia che contiene la classe R_1 che a sua volta contiene la classe R_2, che a sua volta contiene la classe R_3, ossia:

$R_3 \subset R_2 \subset R_1 \subset R_0$

È possibile dimostrare che tale inclusione è propria:



TEOREMA

Per provare l'inclusione propria tra i linguaggi dobbiamo dimostrare i seguenti punti:

1. $R_3 \subset R_2$
2. $R_2 \subset R_1$
3. $R_1 \subset R_0$

DIMOSTRAZIONE DEL PUNTO 1)

Necessitiamo di dimostrare che R_3 è strettamente incluso in R_2, ossia si necessita di dimostrare che:

$\exists L \in R_2 \mid L \notin R_3$

Sia G la grammatica per il linguaggio $L = a^n b^n$ con $n \geq 1$

$G = < \Sigma=\{a, b\} , M=\{S\} , P=\{S \rightarrow aSb \mid ab\} , S >$

- G è di tipo 2? SI in quanto è una grammatica che soddisfa la condizione che α è un metasimbolo, ($\alpha \in M$), β è una forma sentenziale, ossia una sequenza di simboli e/o metasimboli, ad eccezione della parola vuota, ($\beta \in (\Sigma \cup M)^+$)
- G è di tipo 3? NO
  - $S \rightarrow ab$ è della forma $\alpha \rightarrow x$ con $\alpha=S \in M$, $x="ab" \in \Sigma^+$
  - $S \rightarrow aSb$ non è né della forma $\alpha \rightarrow x$, né della forma $\alpha \rightarrow y\beta$

G è quindi una grammatica di tipo 2 ma non di tipo 3 e quindi il linguaggio generabile dalla grammatica G (L) appartiene a R_2 ma non a R_3

DIMOSTRAZIONE DEL PUNTO 2)

Necessitiamo di dimostrare che R_2 è strettamente incluso in R_1, ossia si necessita di dimostrare che:

$\exists L \in R_1 \mid L \notin R_2$

Sia G la grammatica per il linguaggio $L = a^n b^n c^n$ con $n \geq 1$

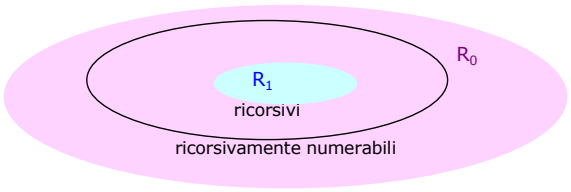
$G = < \Sigma=\{a, b, c\} , M=\{S, B, C\} , P=\{S \rightarrow aSBC|aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\} , S >$

- G è di tipo 1? SI in quanto è una grammatica che soddisfa la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$, le regole di produzione di G, infatti, non consentono di ottenere β mediante la cancellazione di simboli di α.
- G è di tipo 2? NO
  - $CB \rightarrow BC$ non soddisfa la condizione che persista solo UN metasimbolo per α, $\alpha \notin M$ bensì $\alpha \in M^*$
  - $aB \rightarrow ab$ non soddisfa la condizione che α sia metasimbolo, $\alpha \notin M$ bensì $\alpha \in (M \cup \Sigma)^*$

G è quindi una grammatica di tipo 1 ma non di tipo 2 e quindi il linguaggio generabile dalla grammatica G (L) appartiene a R_1 ma non a R_2

DIMOSTRAZIONE DEL PUNTO 3)

Necessitiamo di dimostrare che R_1 è strettamente incluso in R_0, per effettuare questa dimostrazione necessitiamo di dimostrare:



A) i linguaggi in R_1 sono linguaggi ricorsivi

B) i linguaggi in R_0 sono linguaggi ricorsivamente numerabili

C) esiste un linguaggio in R_0 che non è ricorsivo, ossia $\exists$ L $\in R_0$ | L $\notin R_1$

DIMOSTRAZIONE DEL PUNTO A)

RICORDANDO PRIMA CHE: un linguaggio L si dice ricorsivo (o deducibile) se esiste un algoritmo implementato dal programma w tale che, passandogli in input un dato x $\in \{0, 1\}^*$:

$$\begin{cases} F_w(x) = 1 & \text{sse } x \in L \\ F_w(x) = 0 & \text{sse } x \notin L \end{cases}$$

Dobbiamo esibire w per un linguaggio L in R_1, dove w si basa sulla costruzione di un grafo, il quale, viene costruito appositamente sulla parola x che viene fornita in input all'algoritmo w, detto grafo, che cambia a seconda dell'input passatogli, viene denotato con la dicitura GR(x);

GR(x) = (V_x, E_x) dove

V_x sono i vertici del grafo, ossia l'insieme delle forme sentenziali "y" che sono di lunghezza inferiore o alpiù uguale alla lunghezza di x, ovvero tali che |y| ≤ |x|

E_x sono i lati del grafo, ossia l'insieme delle coppie (y, y') tali per cui, nella grammatica G, esista una derivazione in un passo in grado di trasformare y in y', ovvero $\exists$ d | $y \Rightarrow^d y'$

w:
Richiede in input x $\in \Sigma^*$
costruisce GR(x)
esegue If:
If ($\exists$ cammino da S a x) then y = 1
Else y = 0
Return y
output: y

ANALISI DESCRITTIVA DELL'ALGORITMO W:

- Prende in input la parola x $\in \Sigma$
- Costruisce il grafo della parola x ricevuta in input;
- Esegue l'istruzione if rimanendo in attesa di un risultato da parte dell'istruzione.
- fornisce quale output il risultato di IF

ANALISI DESCRITTIVA DELL'ISTRUZIONE IF:

- Opera il seguente test: esiste un cammino che permette da S (assioma) di ottenere la parola x (input passato all'algoritmo)?
- Assegna a "y" (dove y è una variabile binaria, y $\in \{0, 1\}^*$) il valore 1 se esiste un cammino da S a x, assegna ad y il valore 0 se non esiste un cammino da S a x.
- Restituisce y all'algoritmo w

Quando la parola passata in input appartiene al linguaggio questa viene generata dalla grammatica G che genera il linguaggio, quindi x $\in$ L $\Rightarrow$ x $\in$ L(G), ossia $\exists$ y_1, y_2, y_3, …, y_k (forme intermedie sentenziali) tali che:

$S \Rightarrow_G y_1 \Rightarrow_G y_2 \Rightarrow_G y_3 \Rightarrow_G \ldots \Rightarrow_G y_k \Rightarrow_G x$

Se esiste detta derivazione allora nel grafo GR(x) esistono degli archi che mi consentono di visualizzare la derivazione da S a x, ossia esistono i seguenti archi: (S, y_1), (y_1, y_2), (y_2, y_3), (y_3, $y_\ldots$), ($y_\ldots$, y_k), (y_k, x), questo è verificato se e solo se esiste un cammino da S a x nel grafo GR(x).

Dove abbiamo ipotizzato di usare una grammatica di tipo 1?

Quando abbiamo definito x $\in$ L(G), se voglio definire G quale grammatica di tipo 1, nell'effettuare la transizione che mi permette da S di arrivare ad x, ossia, $S \Rightarrow_G y_1 \Rightarrow_G \ldots \Rightarrow_G x$, si necessita di controllare che le regole di produzione che mi generano y_1, y_2, y_3, …, y_k mi consentano solo di far crescere o eguagliare le lunghezze, quindi devo imporre $|y_i| \leq |y_{i+1}| \leq |x|$.

L'imposizione della regola: $|y_i| \leq |y_{i+1}| \leq |x|$ mi garantisce G di tipo 1.

CASO I: TEST DI W PASSANDO IN INPUT $x \in L$

$G = < \Sigma=\{a\} , M=\{S\} , P=\{S \rightarrow aaS \mid aa\} , S >$

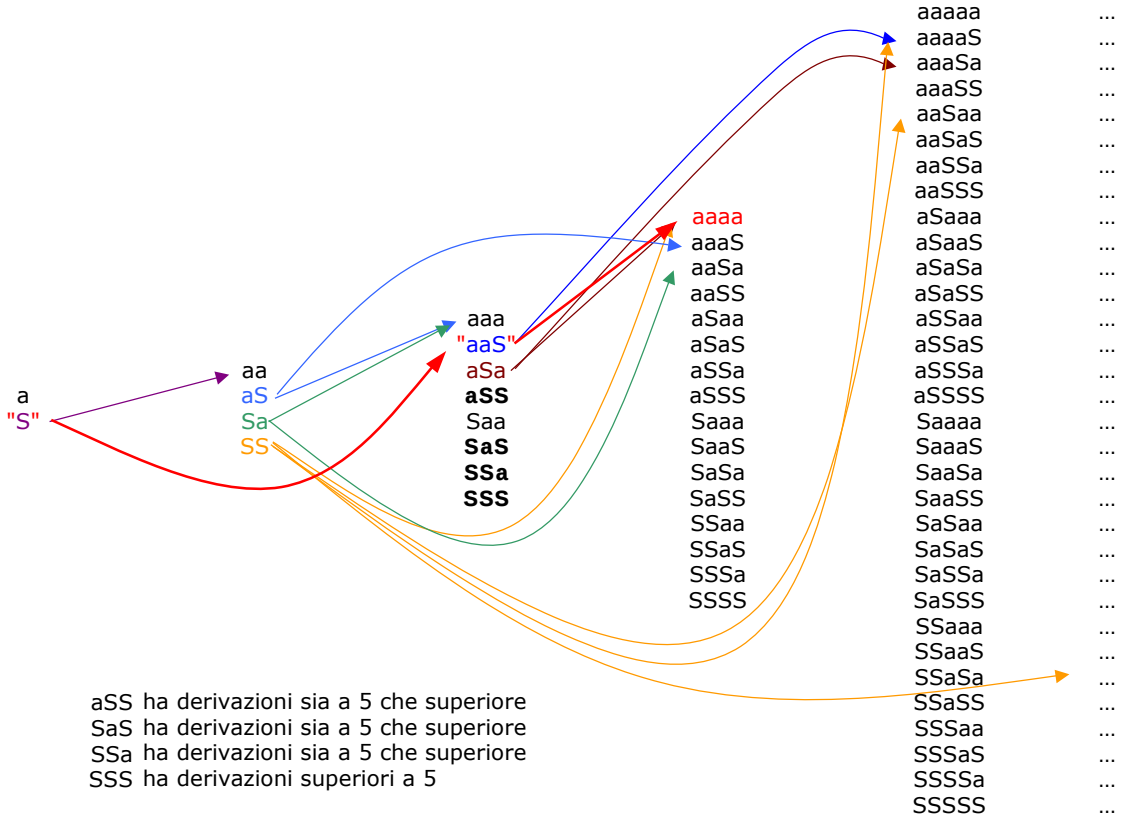
1° passo: verifichiamo che G è una grammatica di tipo 1: SI, infatti tutte le sue regole di produzione non producono una decrescita da una transizione all'altra, inoltre non effettua alcun tipo di cancellazione, infatti:

$S \rightarrow aaS \quad 1 \rightarrow 3, S \rightarrow aa \quad 1 \rightarrow 2$ sono entrambe regole di produzione che consentono solo crescite

2° passo: passiamo all'algoritmo w l'input $x \in \{a\}^*$ dove x = "aaaa"

1 w: riceve in input "aaaa"

2 costruisce il grafo GR(x)



aSS ha derivazioni sia a 5 che superiore
SaS ha derivazioni sia a 5 che superiore
SSa ha derivazioni sia a 5 che superiore
SSS ha derivazioni superiori a 5

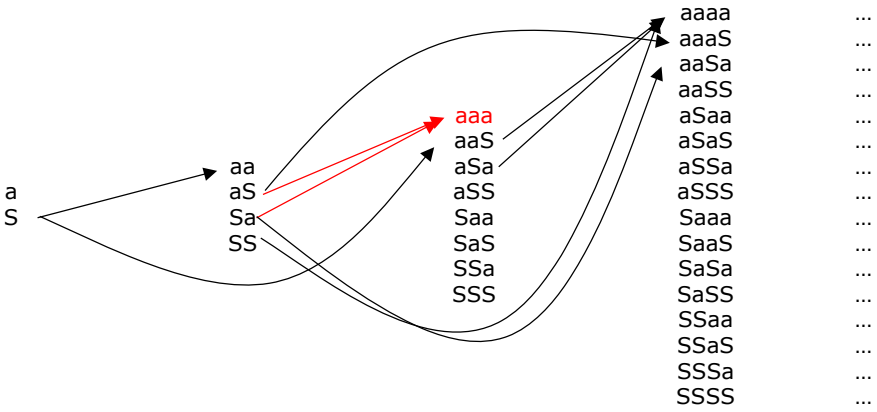
3 $\exists$ cammino da S a x? SI, quello evidenziato in rosso quindi return = 1

CASO II: TEST DI W PASSANDO IN INPUT $x \notin L$

$G = < \Sigma=\{a\} , M=\{S\} , P=\{S \rightarrow aaS \mid aa\} , S >$

1 w: riceve in input "aaa"

2 costruisce il grafo GR(x)



4 $\exists$ cammino da S a x? No, infatti ne aS, ne Sa sono collegati a S quindi return = 0

DIMOSTRAZIONE DEL PUNTO B)

Come abbiamo visto nella definizione di classe R_0, essendo i linguaggi appartenente a detta classe costruibili da una grammatica rientrano nella definizione di linguaggi ricorsivamente numerabili.

DIMOSTRAZIONE DEL PUNTO C)

L'esistenza di un linguaggio ricorsivamente numerabile ma non ricorsivo è stata lungamente discussa e dimostrata a pagina 13 dove si è dimostrata l'esistenza del linguaggio D ricorsivamente numerabile ma non ricorsivo, quindi $\exists D \in R_0 \mid D \notin R_1$

DI CHE TIPO È L SE MI CONCEDE DI GENERARE ε?

Supponiamo di avere un linguaggio L tale che contenga la parola ε al suo interno, di che tipo è L?
L'unico tipo che mi concede di generare ε è il linguaggio di tipo 0, in quanto, a partire dal tipo 1, non è possibile generare parole vuote, per definizione di tipo. Dal tipo 1, infatti, non possono essere definite regole della forma: $A \to \varepsilon$
Questa classificazione, però, non è corretta in quanto, anche linguaggi di tipo 3, ossia linguaggi definiti mediante algoritmo, potrebbero necessitare di contenere la parola vuota al loro interno.
L'aggiunta di ε comporta l'immediato passaggio da grammatica di tipo 3 a grammatica di tipo 0, detto comportamento risulta poco ragionevole.
Necessitiamo quindi di definire una proprietà che:
SE L (CHE NON CONTIENE ε) È DI TIPO K ⇒ ANCHE $L \cup \{\varepsilon\}$ È DI TIPO K
è quindi necessario modificare la classificazione di Chomsky al fine dell'inserimento di detta nuova legge di classificazione.

INSERIMENTO DELLA REGOLA DI PRODUZIONE $S' \to \varepsilon$

Ammettiamo la regola di produzione: $S' \to \varepsilon$
Dove $S' \to \varepsilon$ è la regola di produzione che mi permette di generare la parola vuota

CONDIZIONE "A PATTO CHE"

La concessione dell'inserimento della regola di produzione $S' \to \varepsilon$ viene fatta "a patto che" S' non compaia mai sulla parte destra di alcuna regola di produzione appartenente all'insieme P della grammatica in oggetto, questa condizione mi consente di impedire la creazione di ε durante la derivazione in più passi, bensì mi consente solo di creare la parola vuota.
L'esempio sotto proposto chiarisce la motivazione per la quale viene richiesta l'osservanza della condizione "a patto che" quando viene introdotta la regola di produzione $S' \to \varepsilon$.

S' NUOVO ASSIOMA DI G

Ammettiamo la regola di produzione: $S' \to \varepsilon$
Inseriamo inoltre la regola di produzione $S' \to S$
L'assioma S viene quindi sostituito dal nuovo assioma S'. ricordando che non è possibile utilizzare S' sulla parte destra delle regole di produzione.
ESEMPIO DI NON RISPETTO DELLA CONDIZIONE "A PATTO CHE"

Prendiamo la grammatica G di tipo 1: $G = < \Sigma=\{b, a\} , M=\{S\} , P=\{\dots , aA \to aS, \dots\} , S >$
Ed ammettiamo la regola di produzione $S \to \varepsilon$:
$G = < \Sigma=\{b, a\} , M=\{S\} , P=\{S \to \varepsilon, aA \to aS, \dots\} , S >$
SI NOTI BENE che non è stata rispettata la condizione "a patto che", S infatti compare nella parte destra della regola di produzione: $aA \to aS$
Data la grammatica appena definita, se mi trovassi nella seguente situazione:
$\dots \Rightarrow baA \Rightarrow baS \Rightarrow ba$
l'ultima derivazione comporta la cancellazione di un simbolo all'interno della parola, questo comportamento entra in contrasto con la definizione di grammatica di tipo 1, la quale richiede, espressamente, che vengano vietate cancellazioni durante i diversi passi di derivazione.

GRAMMATICA DI TIPO 0: VERSIONE UNICA

Dato che le grammatiche di tipo 0 prevedono la parola vuota, queste non subiscono variazioni.

GRAMMATICA DI TIPO 1: II VERSIONE

RICORDANDO PRIMA CHE: se ogni regola di produzione $\alpha \to \beta$ soddisfa le seguenti condizioni:
$\alpha \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
$\beta \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
$|\beta| \geq |\alpha|$
allora si parla quindi di grammatica di tipo 1, ossia, in ogni regola di produzione $\alpha \to \beta$ della grammatica, viene soddisfatta la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$.

Aggiungiamo ora alla grammatica di tipo 1 la regola di produzione che ci consente di costruire ε, otterremo la seguente definizione di grammatica di tipo 1:

- $\alpha \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
- $\beta \in (\Sigma \cup M)^+$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M esclusa ε,
- $|\beta| \geq |\alpha|$
- Aggiungere in M il metasimbolo S'
- Aggiungere in P le seguenti regole di produzione: $S' \to \varepsilon \mid S$
- Dato che ∃ la regola di produzione $S' \to \varepsilon$, deve essere rispettata la condizione "a patto che"

Si parla quindi di grammatica di tipo 1 se, in ogni regola di produzione $\alpha \to \beta$ della grammatica, viene soddisfatta la condizione che la lunghezza di β sia maggiore o al più uguale alla lunghezza di α, ossia $|\beta| \geq |\alpha|$, dove, la presenza della regola di produzione $S' \to \varepsilon$ comporta il doversi attenere alle direttive imposte dalla condizione "a patto che".

CONDIZIONE "A PATTO CHE" TROPPO RESTRITTIVA

Per le grammatiche di tipo 2 e 3 ammetto anche le regole di produzione del tipo $A \to \varepsilon$ anche se A non è l'assioma della grammatica, ossia $A \in M$ ma $A \neq S$.

Questa regola si contrappone alle regole imposte per il tipo 1, nonostante ciò, la condizione "a patto che" risulta essere troppo restrittiva per queste due tipologie di grammatiche.

GRAMMATICHE DI TIPO 2: II VERSIONE

Aggiungiamo ora alla grammatica di tipo 2 la regola di produzione del tipo $A \to \varepsilon$, otterremo la seguente definizione di grammatica di tipo 2:

- $\alpha \in M$ è l'insieme dei metasimboli,
- $\beta \in (\Sigma \cup M)^*$ è l'insieme di tutte le parole, simboli terminali o metasimboli di Σ e M,

come si potrà notare $\beta \in (\Sigma \cup M)^+$ è stato trasformato in $\beta \in (\Sigma \cup M)^*$, questo ci consentirà la creazione di regole nella forma $A \to \varepsilon$.

Possiamo ancora dire che $R_2 \subset R_1$? Si in quanto è possibile eliminare tutte le regole di produzione della forma $A \to \varepsilon$ da una grammatica in modo da soddisfare la condizione "a patto che" data per le grammatiche di tipo 1, questa eliminazione non comprometterà il linguaggio che la grammatica G di tipo 2 genera.

Quindi nonostante la classificazione è stata modificata compromettendo apparentemente l'inclusione delle classi, in realtà, grazie alla possibilità di modificare delle regole di produzione è possibile ristabilire la sussistenza delle inclusioni tra le classi.

$R_3 \subset R_2 \subset R_1 \subset R_0$

L'inclusione in rosso quindi è garantita dalla possibilità di trasformare G di tipo 2 in una G' equivalente che soddisfi la condizione "a patto che" imposta dalle Grammatiche di tipo 1, infatti, se L è di tipo 2 con $\varepsilon \in L$, allora $L=L' \cup \{\varepsilon\}$, dove L' è generato da una grammatica con regole del tipo $A \to x$, con $x \in (\Sigma \cup Q)^+$.

ESEMPIO DI ELIMINAZIONE DI REGOLA $A \to \varepsilon$ DALLA GRAMMATICA

Prendiamo la grammatica di tipo 2 che definisce il linguaggio delle parole palindrome, ossia di quelle parole $x=ww^R$ dove w^R è w letta al contrario, ossia se $w = a_1a_2a_3a_4$ allora $w^R = a_4a_3a_2a_1$, esempi di parole palindrome sono: osso, alla, otto, …

Le regole di produzione per la costruzione di parole binarie palindrome sono:

$S \to 0\ S\ 0$

$S \to 1\ S\ 1$

$S \to \varepsilon$

Ottengo quindi la seguente grammatica G di tipo 2:

$G = < \Sigma=\{0, 1\} , M=\{S\} , P=\{S \to 0S0 \mid 1S1 \mid \varepsilon\} , S >$

Vediamo la creazione di una parola palindroma:

$S \Rightarrow^1 0S0 \Rightarrow^2 01S10 \Rightarrow^3 0110$

Come abbiamo detto G è una grammatica di tipo 2 in quanto a sinistra compare sempre e solo un metasimbolo, non possiamo altresì dire che è stata mantenuta l'inclusione rispetto alle grammatiche di tipo 1, questo non rispetto dell'inclusione infatti è data dal fatto che in G compare la regola di produzione $A \to \varepsilon$ senza il rispetto della condizione "a patto che" imposta dalle grammatiche di tipo 1.

Vediamo allora come effettuare le modifiche opportune per mantenere l'inclusione della grammatica G di tipo 2 nelle grammatiche di tipo 1.

Devo eliminare $S \to \varepsilon$ come mi comporto?

Dato che ho	$S \to 0S0$	aggiungo	$S \to 00$
Dato che ho	$S \to 1S1$	aggiungo	$S \to 11$
Elimino	$S \to \varepsilon$		

Quindi $P =\{S \to 0S0 \mid 1S1 \mid \varepsilon\}$ si trasforma in $P =\{S \to 0S0 \mid 00 \mid 1S1 \mid 11\}$

Ora devo apportare un'ulteriore modifica al fine di poter generare la parola vuota, in quanto P è ora predisposta a generare solo parole di tipo $PALINDROME^+$

Si necessita di utilizzare la regola del tipo 1, ossia

- aggiungere in M il metasimbolo S'
- aggiungere in P le seguenti regole di produzione: $S' \to \varepsilon \mid S$
- dato che $\exists$ la regola di produzione $S' \to \varepsilon$, deve essere rispettata la condizione "a patto che"

quindi $P =\{S \to 0S0 \mid 00 \mid 1S1 \mid 11\}$ si trasforma in $P =\{S' \to S, S' \to \varepsilon, S \to 0S0 \mid 00 \mid 1S1 \mid 11\}$

ottengo quindi la grammatica di tipo 2 G' inclusa nelle grammatiche di tipo 1:

$G = < \Sigma=\{0, 1\} , M=\{S, S'\} , P=\{ S' \to S, S' \to \varepsilon, S \to 0S0 \mid 00 \mid 1S1 \mid 11\} , S' >$

ESEMPIO GENERALE DI ELIMINAZIONE DELLE REGOLE

Supponiamo che in una grammatica lineare ci sia la regola di produzione $D \to bbaE$.

Al fine di ottenere una grammatica equivalente avente ogni regola di produzione $\alpha \to \beta$ nella forma

- $A \to \sigma B$
- $A \to \sigma$ con $\sigma \in \Sigma$

dobbiamo sostituire la produzione $D \to bbaE$ con

$D \to bC$
$C \to bF$
$F \to aE$
dove C, F sono due nuovi metasimboli.

ESEMPIO GENERALE DI ELIMINAZIONE DELLE REGOLE

Considera la grammatica con assioma A e con le seguenti regole di produzione:

- $A \to B$
- $B \to C$
- $C \to A$
- $A \to D$
- $B \to aA$
- $D \to a$
- Per eliminare le produzioni del tipo $M \to N$, dobbiamo considerare tutte le derivazioni del tipo $F \Rightarrow_G^* H$. In questo caso abbiamo che $X \Rightarrow_G^* Y$ e $Z \Rightarrow_G^* D$ per $X,Y,Z \in \{A,B,C\}$ e quindi possiamo eliminare le produzioni $A \to B$, $B \to C$, $C \to A$, $A \to D$ inserendo nella grammatica le produzioni $X \to aY$ per $X \in \{A,B,C\}$ e $Y \in \{A,B,C,D\}$, e $Z \to a$ con $Z \in \{A,B,C,D\}$. Lasciamo al lettore il compito di semplificare la grammatica così ottenuta.

GRAMMATICHE DI TIPO 3: II VERSIONE

Aggiungiamo ora alla grammatica di tipo 3 la regola di produzione del tipo $A \to \varepsilon$, otterremo la seguente definizione di grammatica di tipo 3:

- $\alpha \to x$
  - $\alpha \in M$
  - $x \in \Sigma^*$
- $\alpha \to y\beta$
  - $\alpha \in M$
  - $y \in \Sigma^*$
  - $\beta \in M$

come si potrà notare $x \in \Sigma^+$ è stato trasformato in $x \in \Sigma^*$, questo ci consentirà la creazione di regole nella forma $A \to \varepsilon$.

Possiamo ancora dire che $R_3 \subset R_1$? Si in quanto è possibile eliminare tutte le regole di produzione della forma $A \to \varepsilon$ da una grammatica in modo da soddisfare la condizione "a patto che" data per le grammatiche di tipo 1, questa eliminazione non comprometterà il linguaggio che la grammatica G di tipo 3 genera.

TEOREMA DI EQUIVALENZA TRA LE FORME DELLE GRAMMATICHE DI TIPO 3

Cominciamo col richiamare che un linguaggio è di tipo 3 se esiste una grammatica di tipo 3 che lo genera.

Tuttavia è possibile che una grammatica G non di tipo 3 generi un linguaggio di tipo 3; in questo caso deve naturalmente esistere una grammatica G' di tipo 3 equivalente a G. Esistono quindi classi di grammatiche un po' più generali di quelle di tipo 3 che tuttavia generano linguaggi di tipo 3.

Le grammatiche di tipo 3 sono identificate da regole di produzione che assumono una delle seguenti quattro tipologie di forme:

1) $A \to x$ oppure $A \to yB$ (dette lineari destre)
2) $A \to \sigma$ oppure $A \to \sigma B$ oppure $A \to \varepsilon$
3) $A \to \sigma B$ oppure $A \to \varepsilon$
4) $A \to \sigma B$ oppure $A \to \sigma$

dove:

$A, B \in M$

$x, y \in \Sigma^*$

$\sigma \in \Sigma$

come fanno le quattro tipologie di forme ad essere equivalenti? Queste possono essere trasformate da una forma all'altra senza modificare il linguaggio che la grammatica genera.

Consideriamo le grammatiche lineari a destra: una grammatica è detta lineare a destra se le sue produzioni sono del tipo $A \to x$ o $A \to yB$, con A,B metasimboli e x,y parole di simboli terminali (eventualmente ε). Chiaramente una grammatica di tipo 3 è lineare a destra, mentre in generale non è vero il viceversa. Vale tuttavia:

TEOREMA

Sia G una grammatica lineare a destra che genera L. Il metodo dimostrativo consiste nel trasformare la grammatica lineare G in nuove grammatiche equivalenti (che generano cioè lo stesso linguaggio L), fino ad ottenere una grammatica equivalente di tipo 3.

DIMOSTRAZIONE

DATA LA GRAMMATICA LINEARE G:

ogni regola di produzione del tipo $A \to \sigma_1 \ldots \sigma_m B$ (con $m \geq 2$) è eliminabile da una grammatica, ottenendo una grammatica equivalente, a patto dell'introduzione dei nuovi metasimboli:

- M_1
- M_2
- …
- M_{m-1}

E a patto dell'introduzione di nuove regole di produzione:

- $A \to \sigma_1 M_1$
- $M_1 \to \sigma_2 M_2$
- …
- $M_{m-1} \to \sigma_m B$;

DATA LA GRAMMATICA LINEARE G:

ogni regola di produzione del tipo $A \to \sigma_1 \ldots \sigma_m$ (con $m \geq 2$) è anch'essa eliminabile da una grammatica, ottenendo una grammatica equivalente.

sono quindi sempre ottenibili grammatiche equivalenti che contengano solo regole di produzione del tipo:

- $A \to \sigma B$
- $A \to \sigma$
- $A \to \varepsilon$
- $A \to B$.

Questa grammatica tuttavia non è di tipo 3 solo per la presenza di regole del tipo $A \to B$.

Allo scopo di eliminare regole di questo tipo, si considerino tutte le derivazioni del tipo $F \Rightarrow_G^* H$, dove F e H sono metasimboli e si proceda:

- Per ogni regola $A \to \sigma B$, si considerano tutte le coppie di metasimboli F ed H per cui $F \Rightarrow_G^* A$ e $B \Rightarrow_G^* H$ e si aggiungono le regole $F \to \sigma H$; infatti risulta possibile nella grammatica la derivazione $F \Rightarrow_G^* A \Rightarrow_G \sigma B \Rightarrow_G^* H$, che equivale a riscrivere F come σH.
- per ogni regola $A \to \sigma$ oppure $A \to \varepsilon$, si aggiungono le regole $F \to \sigma$ oppure $F \to \varepsilon$, per ogni metasimbolo F per cui $F \Rightarrow_G^* A$; infatti risulta possibile nella grammatica la derivazione $F \Rightarrow_G^* A \Rightarrow_G \sigma$ oppure $F \Rightarrow_G^* A \Rightarrow_G \varepsilon$, che equivale a riscrivere F come σ o come ε.

Possiamo a questo punto eliminare tutte le regole del tipo $A \to B$, ottenendo una grammatica di tipo 3 (G') equivalente a G.

DATA LA GRAMMATICA G:

ogni regola di produzione del tipo $A \to \sigma$, con $\sigma \in \Sigma$ è anch'essa eliminabile da una grammatica, ottenendo una grammatica equivalente contenente solo regole del tipo $A \to \sigma B$, $A \to \varepsilon$.

Allo scopo di eliminare regole del tipo $A \to \sigma$ è sufficiente l'introduzione di un nuovo metasimbolo M e della relativa regola di produzione $M \to \varepsilon$, sostituendo poi ogni regola di produzione del tipo $A \to \sigma$ con una nuova regola di produzione del tipo $A \to \sigma M$. si deduce quindi che se L è di tipo 3, allora è generato da una grammatica di tipo 3 contenenti solo regole del tipo $A \to \sigma B$, $A \to \varepsilon$.

DATA LA GRAMMATICA G:

ogni regola di produzione del tipo $A \to \varepsilon$, è anch'essa eliminabile da una grammatica? In generale no, perché se L è generato da una grammatica contenente solo regole del tipo $A \to \sigma B$, $A \to \sigma$, allora $\varepsilon \notin L$. tuttavia se L è di tipo 3 con $\varepsilon \in L$, allora $L = L' \cup \{\varepsilon\}$, dove L' è un linguaggio di tipo 3 generato da una grammatica contenente solo regole di produzione del tipo $A \to \sigma B$, $A \to \sigma$.

Allo scopo di eliminare regole del tipo $A \to \varepsilon$ è sufficiente l'introduzione, per ogni regola in G del tipo $A \to \sigma B$, dove coesiste anche la regola $B \to \varepsilon$, aggiungere all'insieme delle regole di produzione anche la regola $A \to \sigma$, si deduce quindi che se L è di tipo 3, allora è generato da una grammatica di tipo 3 contenenti solo regole del tipo $A \to \sigma B$, $A \to \sigma$.

ESEMPIO DI TRASFORMAZIONE DALLA FORMA 1) ALLA FORMA 2)

effettuiamo la trasformazione da ($A \to x$ oppure $A \to yB$) a ($A \to \sigma$ oppure $A \to \sigma B$ oppure $A \to \varepsilon$), mediante l'utilizzo del linguaggio $L = (abc)^n$ con $n \geq 1$

$G = < \Sigma=\{a, b, c\} , M=\{S\} , P=\{ S \to abc \mid abcS\} , S >$

$S \to abc$ è una regola di produzione del tipo $A \to x$

$S \to abcS$ è una regola di produzione del tipo $A \to yB$

Devo effettuare la trasformazione di dette due regole al fine di ottenere solo regole della forma 2) ossia regole della forma $A \to \sigma \mid A \to \sigma B \mid A \to \varepsilon$.

$S \to abc \quad \Rightarrow \quad S \to aX$ è una regola di produzione del tipo $A \to \sigma B$
$X \to bc$ è una regola del tipo $A \to x$ e non va bene quindi devo rincominciare
$X \to bc \quad \Rightarrow \quad X \to bC$ è una regola di produzione del tipo $A \to \sigma B$
$C \to c$ è una regola di produzione del tipo $A \to \sigma$
$S \to abcS \quad \Rightarrow \quad S \to aX$ è una regola di produzione del tipo $A \to \sigma B$
$X \to bcS$ è una regola del tipo $A \to yB$ e non va bene quindi devo rincominciare
$X \to bcS \quad \Rightarrow \quad X \to bY$ è una regola di produzione del tipo $A \to \sigma B$
$Y \to cS$ è una regola di produzione del tipo $A \to \sigma B$

Ho ottenuto quindi le seguenti trasformazioni:
$S \to abc \quad \Rightarrow \quad S \to aX, X \to bC, C \to c$
$S \to abcS \quad \Rightarrow \quad S \to aX, X \to bY, Y \to cS$

Ottengo quindi la grammatica G' di tipo 3 equivalente a G ma con regole di produzione nella seconda forma:
$G = < \Sigma=\{a, b, c\} , M=\{S \} , P=\{ S \to aX, X \to bC \mid bY, C \to c, Y \to cS \} , S >$

ESEMPIO DI TRASFORMAZIONE DALLA FORMA 2) ALLA FORMA 3)

effettuiamo la trasformazione da ($A \to \sigma$ oppure $A \to \sigma B$ oppure $A \to \varepsilon$) a ($A \to \sigma B$ oppure $A \to \varepsilon$), mediante l'utilizzo del linguaggio $L = a^*b^+$

$G = < \Sigma=\{a, b\} , M=\{S \} , P=\{ S \to aS \mid aB \mid b \mid bB, B \to bB \mid b\} , S >$
$S \to aS \mid aB \mid bB$ sono regole di produzione del tipo $A \to \sigma B$ quindi sono accettate anche da 3)
$B \to bB$ è una regola di produzione del tipo $A \to \sigma B$ quindi è accettata anche da 3)
$S \to b$ è una regola di produzione del tipo $A \to \sigma$
$B \to b$ è una regola di produzione del tipo $A \to \sigma$

Devo effettuare la trasformazione di dette due regole al fine di ottenere solo regole della forma 3) ossia regole della forma $A \to \sigma B \mid A \to \varepsilon$.

$S \to b \quad \Rightarrow \quad S \to bX$ è una regola di produzione del tipo $A \to \sigma B$
$X \to \varepsilon$ è una regola di produzione del tipo $A \to \varepsilon$
$B \to b \quad \Rightarrow \quad B \to bX$ è una regola di produzione del tipo $A \to \sigma B$
$X \to \varepsilon$ è una regola di produzione del tipo $A \to \varepsilon$

Ho ottenuto quindi le seguenti trasformazioni:
$S \to b \quad \Rightarrow \quad S \to bX, X \to \varepsilon$
$B \to b \quad \Rightarrow \quad B \to bX, X \to \varepsilon$

Ottengo quindi la grammatica G' di tipo 3 equivalente a G ma con regole di produzione nella terza forma:
$G = < \Sigma=\{a, b\} , M=\{S \} , P=\{ S \to aS \mid aB \mid bX \mid bB, B \to bB \mid bX, X \to \varepsilon\} , S >$

ESEMPIO DI TRASFORMAZIONE DALLA FORMA 2) ALLA FORMA 4)

effettuiamo la trasformazione da ($A \to \sigma$ oppure $A \to \sigma B$ oppure $A \to \varepsilon$) a ($A \to \sigma B$ oppure $A \to \sigma$), mediante l'utilizzo del linguaggio $L = a^+b^*c^*$

$G = < \Sigma=\{a, b, c\} , M=\{S\} , P=\{S \to aS \mid a \mid aB, B \to bB \mid \varepsilon \mid cC, C \to cC \mid \varepsilon\} , S >$
$S \to aS \mid aB$ sono regole di produzione del tipo $A \to \sigma B$ quindi sono accettate anche da 4)
$B \to bB \mid cC$ sono regole di produzione del tipo $A \to \sigma B$ quindi sono accettate anche da 4)
$C \to cC$ è una regola di produzione del tipo $A \to \sigma B$ quindi è accettata anche da 4)
$S \to a$ è una regola di produzione del tipo $A \to \sigma$ quindi è accettata anche da 4)
$B \to \varepsilon$ è una regola di produzione del tipo $A \to \varepsilon$
$C \to \varepsilon$ è una regola di produzione del tipo $A \to \varepsilon$

Devo effettuare la trasformazione di dette due regole al fine di ottenere solo regole della forma 4) ossia regole della forma $A \to \sigma B \mid A \to \sigma$.

$B \to \varepsilon \quad \Rightarrow$ procedo cercando tutte le regole di produzione che contengono il metasimbolo B nella parte destra della regola di produzione e per ciascuna regola contenente $X \to xB$ pongo una nuova regola aggiuntiva $X \to x$, ossia
$S \to aB$ contiene B $\Rightarrow$ $S \to a$ è del tipo $A \to \sigma$ quindi accettata da 4)
$B \to bB$ contiene B $\Rightarrow$ $B \to b$ è del tipo $A \to \sigma$ quindi accettata da 4)
$C \to \varepsilon \quad \Rightarrow$ procedo cercando tutte le regole di produzione che contengono il metasimbolo C nella parte destra della regola di produzione e per ciascuna regola contenente $X \to xC$ pongo una nuova regola aggiuntiva $X \to x$, ossia
$B \to cC$ contiene C $\Rightarrow$ $B \to c$ è del tipo $A \to \sigma$ quindi accettata da 4)
$C \to cC$ contiene C $\Rightarrow$ $C \to c$ è del tipo $A \to \sigma$ quindi accettata da 4)

Ho ottenuto quindi le seguenti trasformazioni:
$B \to \varepsilon \quad \Rightarrow \quad S \to a, B \to b$
$C \to \varepsilon \quad \Rightarrow \quad B \to c, C \to c$

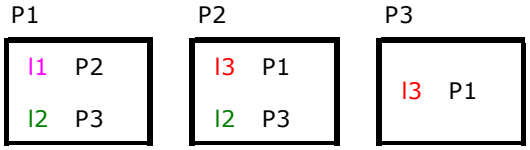
Ottengo quindi la grammatica G' di tipo 3 equivalente a G ma con regole di produzione nella quarta forma:
$G = < \Sigma=\{a, b, c\} , M=\{S\} , P=\{S \to aS \mid a \mid aB, B \to bB \mid b \mid c, C \to cC \mid c\} , S >$

AUTOMI A STATI FINITI

ESEMPIO INTRODUTTIVO:

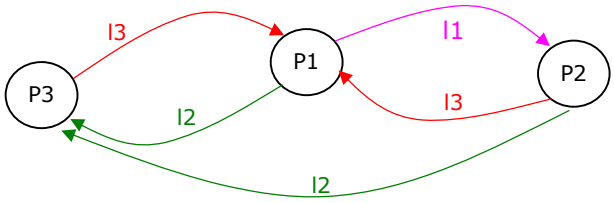
vediamo come modellare le interfacce di pagine web con gli automi, supponiamo di avere:

- l1 →link che consente di andare a pagina 2
- l2 →link che consente di andare a pagina 3
- l3 →link consente di andare a pagina 1



- P1 → pagina web 1 dotata di due link: l1 e l2;
- P2 → pagina web 2 dotata di due link: l2 e l3;
- P3 → pagina web 3 dotata di un solo link: l3;

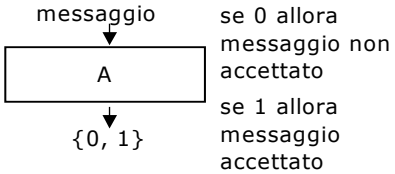
RAPPRESENTAZIONE GRAFICA MEDIANTE IL DIAGRAMMA DEGLI STATI DELL'AUTOMA:



ciascuno stato (P1, P2 e P3), nella rappresentazione grafica, deve essere inserito all'interno di un cerchio, per evidenziare, invece, i passaggi da uno stato all'altro vengono utilizzate delle frecce, ciascuna di queste è legata ad un evento, nel nostro esempio, ciascuna freccia identifica un click effettuato su un determinato link. Questo è l'automa che schematizza mediante un diagramma a stati il movimento tra 3 diverse pagine web, dall'osservazione della rappresentazione grafica viene subito posto in risalto la mancanza di passaggi diretti che consentano di muoversi dalla pagina 3 (P3) alla pagina 2 (P2) mediante un link diretto, l'unico modo, infatti, per raggiungere P2, partendo da P3, è effettuare precedentemente un passaggio attraverso P1.

COS'È UN AUTOMA?

vogliamo introdurre un sistema che modelli un semplice meccanismo di interazione con una "scatola nera", dotata di un ingresso e di una uscita. Supponiamo che tale sistema possa ricevere in ingresso messaggi, dati da un alfabeto finito $S = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, e produrre in uscita un valore in $\{0, 1\}$ che può essere osservato. Riassumendo un automa è un sistema schematizzabile mediante una scatola la quale acquisisce in input un messaggio su un certo alfabeto Σ e fornisce in output un risultato esprimibile sotto forma di 0/1 dove 1 significa che il messaggio è stato accettato e riconosciuto dall'automa, 0 significa che il messaggio non è stato accettato.



ESPERIMENTO: è il risultato che fornisce l'automa se gli viene fornito un singolo messaggio, consiste nei seguenti due passi:

- Viene presentata una sequenza di messaggi, descritta da una parola $w \in \Sigma^*$
- Al termine, viene effettuata una osservazione: se il risultato è 1 la parola viene accettata, altrimenti, nel caso in cui il risultato è 0 la parola viene respinta

COMPORTAMENTO: Poiché possiamo accedere al sistema solo attraverso esperimenti, il comportamento del sistema è descritto dall'insieme di parole accettate; questo sistema è quindi visto come riconoscitore di linguaggi.

CARATTERISTICHE PRINCIPALI INTERNE DELL'AUTOMA

- Un automa si compone di stati, viene indicato con "Q" l'insieme degli stati che costituiscono l'automa;
- In ogni istante l'automa persiste in un preciso stato $q \in Q$ e questo stato può essere modificato solo attraverso un messaggio $\sigma \in \Sigma$ inviato in ingresso;
- inizialmente l'automa deve trovarsi nello stato che prende il nome di "stato iniziale" e che si indica con "q_0", dove $q_0 \in Q$, altrimenti, se l'automa non viene inizializzato, può sviluppare il cammino riconoscitivo di una parola da uno degli stati di cui si compone, si ottengono, quindi, risposte differenti a medesimi messaggi in ingresso in base allo stato in cui è fermo l'automa quando viene fornito il messaggio, ecco la motivazione per la quale si necessita dell'inizializzazione in q_0;

RICORDANDO PRIMA CHE: dati due insiemi non vuoti A e B, chiamiamo funzione di A in B ($f: A \to B$) una qualunque relazione che associa ad ogni elemento $x \in A$ uno e un solo elemento $y \in B$, e la funzione in simboli si può indicare in maniera equivalente con: $y = f(x)$ o $f: x \to y$.
y prende il nome di immagine di x in f, A prende il nome di dominio della funzione
f(A) ossia l'insieme delle immagini della funzione prendono il nome di codominio

- Ad ogni mossa l'automa affettua la lettura di un simbolo e il relativo cambiamento di stato, la legge che descrive la modifica di stato interno causata dall'arrivo di un messaggio è data dalla "funzione di transizione" o anche detta "funzione di stato prossimo", viene identificata con il simbolo $\delta: Q \times \Sigma \to Q$. se il sistema si trova nello stato q ed arriva il messaggio σ, il sistema passa nello stato $\delta(q, \sigma)$ e si dice che "$\delta(q, \sigma)$ è lo stato in cui arriva il sistema, inizializzata con q, dopo aver applicato l'azione corrispondente alla lettura del messaggio σ", $\delta(q, \sigma)$ è il condominio della funzione di transizione δ che definisce come cambia lo stato dell'automa in funzione del simbolo σ letto partendo dallo stato q;
- La risposta dell'automa o anche detta "osservazione sul sistema", dopo la lettura del messaggio, è descritta da una "funzione di uscita $\lambda: Q \to \{0, 1\}$, detta funzione di uscita è strettamente dipendente allo stato in cui si trova l'automa dopo la lettura del messaggio, se il sistema è nello stato q, il risultato dell'osservazione è $\lambda(q)$;

$$\lambda(q) \begin{cases} 1 & \text{se q è uno stato che serve per descrivere l'uscita "1"} \\ 0 & \text{se q è uno stato che serve per descrivere l'uscita "0"} \end{cases}$$

AUTOMA A STATI

Un automa a stati "A" è un sistema A descritto dalla quintupla di elementi:
$A = <\Sigma, Q, \delta, q_0, \lambda/F>$, dove Q è un insieme di stati, se Q è un insieme finito l'automa prenderà il nome di automa a stati finiti, Σ è un alfabeto finito, $\delta: Q \times \Sigma \to Q$ è la funzione di transizione, $q_0 \in Q$ è lo stato iniziale, $F \subseteq Q$ è l'insieme degli stati finali che definisce una funzione $\lambda: Q \to \{0, 1\}$, dove $\lambda(q) = 1$ se $q \in F$, viceversa = 0 se $q \notin F$, vediamo ora ciascun componente della quintupla in dettaglio:

- Σ, alfabeto sul quale l'automa lavora, prende anche il nome di "alfabeto di input";
- Q, insieme degli stato di cui l'automa A si compone, se Q è un insieme finito (ossia composto da un numero finito di elementi) allora l'automa prende il nome di automa a stati finiti;
- δ, funzione di transizione che deve essere descritta:
  - mediante l'indicazione del dominio e codominio, ossia: $\delta: Q \times \Sigma \to Q$;
  - solo nel caso in cui sia il dominio che il codominio sono insiemi finiti allora δ è rappresentabile mediante tabella, dove nelle colonne vengono indicati i $q_i \in Q$ (con i che va da 0 ad h) e in riga vengono indicati i $\sigma_j \in \Sigma$ (con j che va da 1 a k), all'intersezione tra q_i e σ_j viene inserito lo stato che raggiunge l'automa che si trova nello stato q_i e legge il simbolo σ_j

δ	q_0	q_1	q_2	...	q_i	...	q_h
σ_1	$\delta(q_0, \sigma_1)$	.	.	...	.	...	.
σ_2	.	.	$\delta(q_2, \sigma_2)$	...	.	...	.
σ_3	.	.	.	...	.	...	.
...	...	...	...	...	...	...	...
σ_j	.	.	.	...	$\delta(q_i, \sigma_j)$	...	.
...	...	...	...	...	...	...	...
σ_k	.	$\delta(q_1, \sigma_k)$					

- Q_0, indica lo stato iniziale dell'automa;
- FUNZIONE DI USCITA, la funzione di uscita permette due descrizioni tra loro ambivalenti con le quali rappresentarla:
  - attraverso la funzione λ, ossia $\lambda: Q \to \{0, 1\}$;
  - attraverso la descrizione di tutti gli stati che, se raggiunti, mi consentono di ottenere uscita = 1, in questo caso l'uscita viene indicata con "F", dove $F \subseteq Q$;

AUTOMA COME RICONOSCITORE DI LINGUAGGI

Come possiamo descrivere il linguaggio riconosciuto dall'automa A, ossia quell'insieme di parole che soddisfano la proprietà δ dell'automa? la funzione δ, la quale indica il comportamento dell'automa alla lettura del carattere σ quando si trovava nello stato q, ossia δ: Q x Σ→Q, può essere univocamente estesa non più alla lettura di caratteri $\sigma \in \Sigma$ ma ad intere parole $w \in \Sigma^*$, ottenendo quindi la funzione estesa δ*, dove δ*: Q x Σ* → Q:

δ*(q_0, w): è il condominio della funzione di transizione δ* che definisce come cambia lo stato dell'automa in funzione della lettura della parola $w \in \Sigma^*$ partendo dallo stato iniziale q_0, lo star è omettibile, nonostante ciò non crea problemi nella distinzione tra δ e δ* in quanto le due funzioni operano su due insiemi completamente diversi: $\delta: Q \times \Sigma \to Q$ $\delta^* : Q \times \Sigma^* \to Q$

Descrizione di δ* in maniera ricorsiva:

- $\forall q \in Q, \delta^*(q, \varepsilon) = q$, ossia, lo stato in cui arriva il sistema, inizializzato con q, dopo aver applicato l'azione corrispondente alla lettura di nessun messaggio (ε) è uguale allo stato di inizializzazione;
- $\forall w \in \Sigma^*, \delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$, ossia, lo stato in cui arriva il sistema, inizializzato con q, dopo aver applicato le azioni corrispondenti alla sequenza wσ di messaggi [δ*(q, wσ)] è equivalente allo stato in cui arriva il sistema, inizializzato con lo stato in cui è arrivato il sistema inizializzato con q, dopo aver applicato le azioni corrispondenti alla sequenza w di messaggi [δ*(q, w)=q1], dopo aver applicato l'azione corrispondente al simbolo σ [δ(q1, σ)];

L(A): LINGUAGGI RICONOSCIUTO DALL'AUTOMA A

Una volta ricavata δ* si può esprimere il linguaggio L(A) riconosciuto dall'automa A così definito:

- $L(A) = \{w \in \Sigma^* \mid \lambda(\delta^*(q_0, w)) = 1\}$ possiamo definire il linguaggio L(A) come l'insieme di tutte le parole definite su Σ tali che, se fornite all'automa come esperimenti, l'automa, partendo dallo stato iniziale e analizzando le parole fornite arrivi sempre in uno stato legato alla funzione d'uscita λ=1
- $L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$ possiamo definire il linguaggio L(A) come l'insieme di tutte le parole definite su Σ tali che, se fornite alla funzione di transizione, insieme allo stato di partenza q_0, la funzione di transizione porta in uno stato appartenente all'insieme degli stati che consentono di ottenere uscita 1
- Informalmente L(A) può anche essere descritto con il cammino nel diagramma degli stati sotto definito, gli elementi di L(A) sono definiti a partire dall'insieme dei cammini che nel grafo di transizione degli stati sono inizialmente etichettati con q_0 e terminano in uno stato finale ∈ F ed evidenziato da un doppio cerchio

RAPPRESENTAZIONE GRAFICA MEDIANTE IL DIAGRAMMA DEGLI STATI DELL'AUTOMA

Un automa a stati finiti può essere ulteriormente rappresentato come diagramma degli stati, cioè come un grafo orientato in cui i vertici (indicati da delle circonferenze) rappresentano gli stati e i lati (indicati da archi etichettati con simboli ∈ Σ) rappresentano le possibili transizioni tra stati. Lo stato iniziale viene indicato con una freccia in ingresso, mentre gli stati finali vengono indicati con una doppia circonferenza, qui di seguito è l'elenco delle simbologie utili per il disegno del diagramma degli stati:


rappresenta uno stato dell'automa, qi


rappresenta lo stato iniziale dell'automa, q0


rappresenta lo stato finale dell'automa ossia quei qj che appartengono all'insieme F

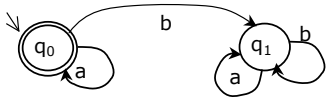

rappresenta le transizioni tra due stati alla lettura di un simbolo del messaggio di input

ESERCIZIO

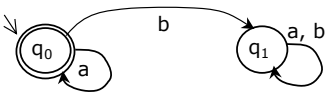
	q_0	q_1
a	q_0	q_1
b	q_1	q_1

$A = <\Sigma = \{a, b\}, Q = \{q_0, q_1\}, \delta: Q \times \Sigma \to Q =$, q_0, $F = \{q_0\}>$

Rappresentazione grafica mediante il diagramma degli stati dell'automa:



analogamente rappresentabile con un'unica freccia contraddistinta dalla dicitura "a, b":



vediamo ora di individuare se stringhe appartenenti all'alfabeto Σ sono accettate dall'automa:

"aba" è accettata dall'automa? no, infatti: $q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_1$ questo è il cammino che induce la lettura della stringa "aba" e termina nello stato $q_1 \notin F$ quindi non è riconosciuta,

"aaa" è accettata dall'automa? SI, $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0$, dato che $q_0 \in F \to$ "aaa" è una stringa riconosciuta

"aab" è accettata dall'automa? no, infatti: $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1$ questo è il cammino che induce la lettura della stringa "aab" e termina nello stato $q_1 \notin F$ quindi non è riconosciuta

"ε" è accettata dall'automa? SI, q_0, non vengono effettuate trasformazioni quindi l'automa resta nello stato q_0 il quale è uno stato finale quindi viene accettato

qual è il linguaggio L(A)? a*

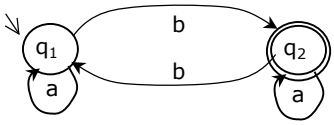
ESERCIZIO

$A = <\Sigma = \{a, b\}, Q = \{q_1, q_2\}, \delta: Q \times \Sigma \to Q = \{q_1, q_2\} \times \{a, b\} \to \{q_1, q_2\} =$

	q_1	q_2
a	q_1	q_2
b	q_2	q_1

$, q_1, F = \{q_2\}>$

Il suo diagramma degli stati è il seguente:



si osservi che ogni parola $w \in \Sigma^*$ induce nel diagramma degli stati un cammino, dallo stato iniziale q_1 ad uno stato q, così che il linguaggio accettato dall'automa è dato dalle parole che inducono cammini che portano dallo stato iniziale in uno stato finale.

vediamo ora di individuare se stringhe appartenenti all'alfabeto Σ sono accettate dall'automa:

"abb" è accettata dall'automa? no, infatti: $q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_1$ questo è il cammino che induce la lettura della stringa "abb" e termina nello stato $q_1 \notin F$ quindi non è riconosciuta,

"aba" è accettata dall'automa? SI, $q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_2$, dato che $q_2 \in F \to$ "aba" è una stringa riconosciuta

"ababa" è accettata? no, infatti: $q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_2 \xrightarrow{b} q_1 \xrightarrow{a} q_1$, questo è il cammino che induce la lettura della stringa "ababa" e termina nello stato $q_1 \notin F$ quindi non è riconosciuta

"ε" è accettata dall'automa? no, infatti da q_1, non vengono effettuate trasformazioni quindi l'automa resta nello stato q_1 il quale non è uno stato finale quindi non viene accettata;

qual è il linguaggio L(A)? **???**

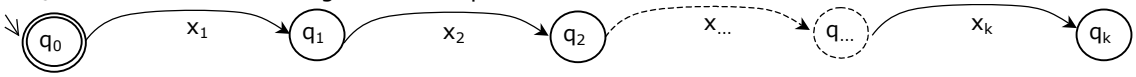
AUTOMI EQUIVALENTI

due automi si dicono equivalenti se riconoscono lo stesso linguaggio: dato l'automa A e l'automa B, A e B sono equivalenti $\leftrightarrow$ L(A) =L(B)

LO STATO TRAPPOLA

Vediamo la creazione di un'automa che riconosce una sola parola:

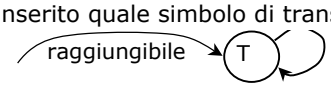
$x = x_1x_2x_3...x_k$ dove k indica la lunghezza della parola che deve essere un numero finito



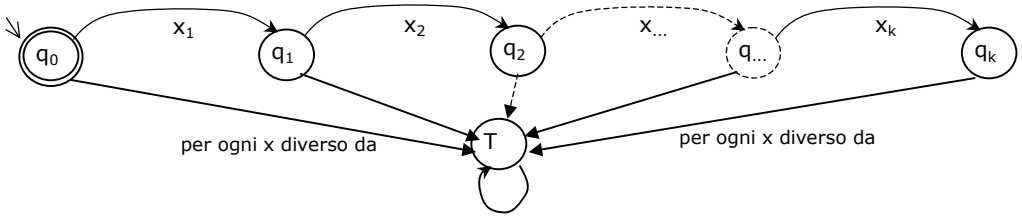
dove vanno inseriti tutte le transazioni derivanti dall'inserimento di σ allo stato q-esimo differente dal simbolo xj richiesto per detto stato?

Viene creato uno stato trappola (T) il quale è così definito:

- Se dallo stato q_i non si specifica una tipologia di transazione, questa, se accade, porta automaticamente allo stato trappola;
- Lo stato trappola è uno stato raggiungibile;
- Lo stato trappola non è mai uno stato finale;
- Lo stato trappola è uno stato che non consente di ritornare all'albero, non è quindi possibile uscire in alcun modo dallo stato T; ovviamente per non poter uscire dallo stato trappola questo deve essere creato con un arco che da T ricade in T $\forall \sigma \in \Sigma$ inserito quale simbolo di transazione.



l'automa sopra definito acquista quindi le seguenti sembianze, dopo l'inserimento dello stato trappola T:



AUTOMA OSSERVABILE

Dato un automa a stati A = <Σ , Q, δ*: Q x Σ* → Q, q_0, F>, il comportamento di A è il linguaggio L(A) = {w ∈ Σ* | λ(δ*(q_0, w)) = 1} = {w ∈ Σ* | δ*(q_0, w) ∈ F}

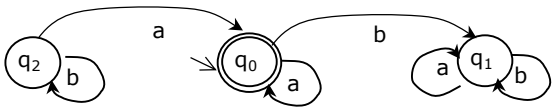
Essenzialmente il comportamento è ottenuto dai risultati degli esperimento sull'automa, poiché λ(δ*(q_0, w)) è il risultato di una osservazione dopo che l'automa ha processato la sequenza di messaggi descritta dalla parola w.

Gli automi osservabili sono automi in cui, ogni stato, è osservabile, dove uno stato q ∈ Q si definisce osservabile se esiste una parola w ∈ Σ* che permette di raggiungere lo stato q partendo dallo stato q_0, ossia:

q osservabile ↔ ∃ w ∈ Σ* | δ*(q_0, w) = q

gli stati non osservabili in un automa sono irrilevanti per quanto riguarda il riconoscimento e possono essere tranquillamente soppressi (cancellati) dall'automa, rendendo l'automa osservabile. Per verificare se uno stato q_k è osservabile o meno bisogna verificare quali sono le frecce che entrano nello stato, se la freccia entrante nello stato q_k proviene da uno stato q_j osservabile allora anche q_k risulterà essere uno stato osservabile, se, invece, lo stato q_k è isolato o le frecce che conducono nello stato q_k arrivano tutte da stati non osservabili, allora anche q_k non è osservabile.

ESEMPIO DI AUTOMA CON STATI NON OSSERVABILI

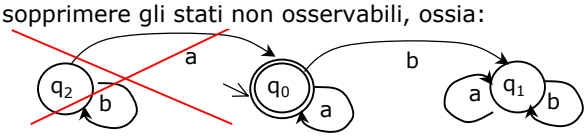


q_0 è uno stato osservabile? Si, ∃ w = "ε" ∈ Σ* | δ*(q_0, ε) = q_0

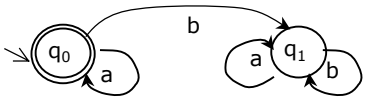
q_1 è uno stato osservabile? Si, ∃ w = "b" ∈ Σ* | δ*(q_0, b) = q_1

q_2 è uno stato osservabile? NO, non esiste una parola w ∈ Σ*|δ*(q_0, w)=q_2 infatti ∀ w ∈ Σ*|δ*(q_0, w)≠q_2

si tratta quindi di un automa non osservabile in quanto contenente lo stato q_2 che risulta non essere osservabile, come facciamo a rendere il nostro automa osservabile? Come detto prima è sufficiente sopprimere gli stati non osservabili, ossia:



ottenendo quindi il seguente automa osservabile



riverifichiamo l'osservabilità del nuovo automa ottenuto dopo la soppressione dello stato q_2:

q_0 è uno stato osservabile? Si, ∃ w = "ε" ∈ Σ* | δ*(q_0, ε) = q_0

q_1 è uno stato osservabile? Si, ∃ w = "b" ∈ Σ* | δ*(q_0, b) = q_1

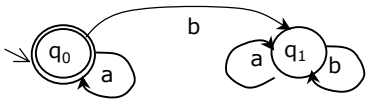
AUTOMA COMPLETAMENTE SPECIFICATO

Gli automi completamente specificati sono automi in cui, per ogni stato, esistono tanti archi uscenti quanti sono i simboli appartenenti a Σ, quindi ciascuno stato q_i ha tutti e solo archi con σ_i ∈ Σ

ESEMPIO DI AUTOMA COMPLETAMENTE SPECIFICATO

	q_0	q_1
a	q_0	q_1
b	q_1	q_1

A = <Σ = {a, b}, Q = {q_0, q_1}, δ: , q_0, F = {q_0}>



q_0 è completamente specificato? Si, ∃ sia l'arco "a" che l'arco "b" ricordando che Σ = {a, b}

q_1 è completamente specificato? Si, ∃ sia l'arco "a" che l'arco "b" ricordando che Σ = {a, b}

AUTOMA NON COMPLETAMENTE SPECIFICATO

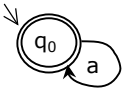
Gli automi non completamente specificati sono automi in cui, almeno uno stato, non esistono tanti archi uscenti quanti sono i simboli appartenenti a Σ, quindi ∃ almeno uno stato q_k il quale ha un numero di archi uscenti da detto stato inferiore al numero di simboli appartenenti all'alfabeto di riferimento Σ.

In questo caso si sottintende l'esistenza di uno stato trappola a cui l'automa farà riferimento quando allo stato q_k (stato non contenente l'arco di transizione con riconoscimento del simbolo σi) viene chiesto di effettuare la transizione dettata dal simbolo σi.

ESEMPIO DI AUTOMA NON COMPLETAMENTE SPECIFICATO

	q_0	q_1
a	q_0	q_1
b	q_1	q_1

A = <Σ = {a, b}, Q = {q_0, q_1}, δ: , q_0, F = {q_0}>



q_0 è completamente specificato? NO, ∃ l'arco "a" ma non è presente l'arco "b" ricordando che Σ = {a, b}

qual'ora avvenisse la richiesta a q_0 di effettuare la transizione "b" questo raggiungerà lo stato trappola T.

STATI DISTINGUIBILI ($\not\approx$)

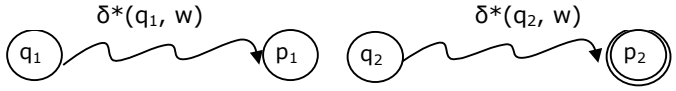
Intuitivamente si potrebbe dire che due stati sono differenti quando mi consentono di fare due cose diverse, ma cosa significa che uno stato "mi consente di operare una cosa diversa" di un altro stato?

Dato un automa a stati $A = <\Sigma , Q, \delta^*: Q \times \Sigma^* \rightarrow Q, q_0, F>$, per definizione due stati $q_1, q_2 \in Q$ si dicono distinguibili e si scrive $q_1 \not\approx q_2$ se e solo se è possibile distinguere con un esperimento il comportamento dell'automa inizializzato con q_1 da quello inizializzato con q_2, ossia,

$\exists\ w \in \Sigma^*$ si ha che $\lambda (\delta^*(q_1, w)) \neq \lambda(\delta^*(q_2, w))$

dire che $q_1 \not\approx q_2$ significa asserire che esiste almeno una parola w in Σ^* tale che, lo stato raggiunto nell'automa, partendo da q_1, leggendo detta parola w sia diverso dallo stato raggiunto nell'automa, partendo da q_2, leggendo la medesima parola w, questo deve accadere per almeno una parola appartenente a Σ^*.

forniamo all'automa A quale input la parola $w \in \Sigma^*$



se $q_1 \not\approx q_2 \Rightarrow$ se p_1 è uno stato che fornisce uscita 0 allora p_2 deve fornire uscita 1, viceversa se p_1 fornisce uscita 1 allora p_2 deve fornire uscita 0, gli stati sono quindi contrapposti, infatti,

$q_1 \not\approx q_2 \Rightarrow$ p_1 finale $\Rightarrow$ p_2 non finale viceversa p_1 non finale $\Rightarrow$ p_2 finale

STATI INDISTINGUIBILI ($\approx$)

Intuitivamente si potrebbe dire che due stati sono equivalenti quando mi consentono di fare la stessa cosa, ma cosa significa che uno stato "consente di fare la stessa cosa" di un altro stato?

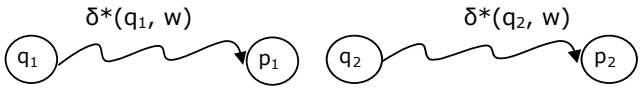
Dato un automa a stati $A = <\Sigma , Q, \delta^*: Q \times \Sigma^* \rightarrow Q, q_0, F>$, per definizione due stati $q_1, q_2 \in Q$ si dicono indistinguibili e si scrive $q_1 \approx q_2$ se e solo se

$\forall\ w \in \Sigma^*$ si ha che $\lambda (\delta^*(q_1, w)) = \lambda(\delta^*(q_2, w))$

Se penso al comportamento dell'automa inizializzata con q_1, ossia l'insieme di parole che l'automa accetta, detto comportamento è analogo se l'automa viene inizializzata con q_2, ossia dire che $q_1 \approx q_2$ significa asserire che lo stato raggiunto nell'automa, partendo da q_1, leggendo una qualsiasi parola w deve essere obbligatoriamente uguale allo stato raggiunto nell'automa, partendo da q_2, leggendo la medesima parola w, questo deve accadere per tutte le parole appartenenti a Σ^*.

Ovviamente due stati sono indistinguibili se e solo se non sono distinguibili.

forniamo all'automa A quale input la parola $w \in \Sigma^*$



se $q_1 \approx q_2 \Rightarrow$ se p_1 è uno stato che fornisce uscita 1 anche p_2 deve fornire uscita 1, viceversa se p_1 fornisce uscita 0 allora anche p_2 deve fornire uscita 0.

$q_1 \approx q_2 \Rightarrow$ p_1 finale $\Rightarrow$ p_2 finale viceversa p_1 non finale $\Rightarrow$ p_2 non finale

i due stati devono quindi essere entrambi, contemporaneamente, finali o entrambi, non finali.

Se $q_1 \approx q_2 \Rightarrow$ uno dei due può essere eliminato in quanto è uno stato superfluo, l'analisi del comportamento delle parole nei due stati è analoga quindi uno dei due viene cancellato.

PROPRIETÀ DI $\approx \subseteq Q \times Q$

RICORDANDO PRIMA CHE: se φ è una relazione sull'insieme A, ossia $\varphi \subseteq A \times A$, si dice che φ è una relazione di equivalenza se φ gode di tutte e tre le seguenti proprietà:

riflessiva: $a\ \varphi\ a$ $\forall\ a \in A$

simmetrica: $a\ \varphi\ b \Rightarrow b\ \varphi\ a$ $\forall\ a,b \in A$

transitiva: $a\ \varphi\ b, b\ \varphi\ c \Rightarrow a\ \varphi\ c$ $\forall\ a,b,c \in A$

1. RIFLESSIVA: $\forall\ q \in Q$ $q \approx q$
2. SIMMETRICA: $\forall\ q, p \in Q$ $q \approx p \Rightarrow p \approx q$
3. TRANSITIVA: $\forall\ q, p, s \in Q$ $q \approx p$ e $p \approx s \Rightarrow q \approx s$
4. INDISTINGUIBILITÀ $q \approx p \Rightarrow \forall\ w \in \Sigma^*, \delta^*(q, w) = \delta^*(p, w)$

è immediato dalla definizione dimostrare che la relazione di indistinguibilità è una relazione di equivalenza in quanto sono definite le proprietà 1., 2., 3,. Verifichiamo quindi solo la proprietà 4. che consente alla relazione di indistinguibilità di prendere il nome di relazione di congruenza.

Dimostrazione della proprietà 4.

Supponiamo che $q_1, q_2 \in Q \mid q_1 \approx q_2$. per parole $w, x \in S^*$ vale che:

$\delta^*(\delta^*(q_1, x), w) = \delta^*(q_1, wx)$ ne segue che $\lambda(\delta^*(\delta^*(q_1, x), w)) = \lambda(\delta^*(q_1, wx))$

dato che $q1 \approx q2 \Rightarrow \lambda(\delta^*(q_1, wx)) = \lambda(\delta^*(q_2, wx)) = \lambda(\delta^*(\delta^*(q_2, x), w))$

ne segue $\delta^*(\delta^*(q_2, x), w) = \delta^*(q_2, wx)$

questo prova quindi $\delta^*(q_1, wx) = \delta^*(q_2, wx)$

PARTIZIONE DELL'INSIEME Q IN CLASSI DI EQUIVALENZA

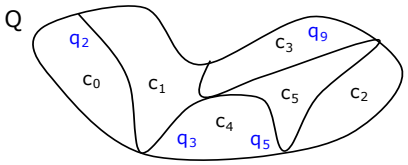
RICORDANDO PRIMA CHE: si dice partizione di un insieme A ≠ Ø ogni collezione {Xi | i ∈ I} di sottoinsiemi non vuoti X, di A tali che:

- $X_i \cap X_j = \emptyset$ per $i \neq j$ # ciascuna partizione è disgiunta dalle altre
- $\cup X_i = A$ # l'unione di tutte le partizioni è l'insieme di partenza

RICORDANDO PRIMA CHE: Dato l'insieme A e la relazione d'equivalenza R ($R \subseteq A \times A$),
$\forall a \in A$ si può considerare la classe di equivalenza individuata da a: $[a]_R = \{b \in A \mid b\ R\ a\}$
si può quindi dire che la classe di equivalenza è l'insieme degli elementi $b \in A$ che sono equivalenti ad a, il simbolo che identifica la classe in relazione di equivalenza è il seguente:[a]. le classi di equivalenza di un insieme A costituiscono una partizione di A, cioè una collezione di sottoinsiemi di A che sono disgiunti e la loro unione da tutto A.

è possibile effettuare la partizione di Q in classi di equivalenza, dove ciascuna partizione è disgiunta dalle altre ($c_i \cap c_j = \emptyset\ \forall$ i, j), l'unione di tutte le partizioni deve risultare essere l'insieme Q di partenza. La classe di equivalenza in cui si trova lo stato q prenderà il nome di $[q]_{\approx}$,

ESEMPIO DI CLASSI DI EQUIVALENZA:



classe di equivalenza di $q_5 = [q_5]_{\approx} = c_4$; classe di equivalenza di $q_3 = [q_3]_{\approx} = c_4$;
classe di equivalenza di $q_2 = [q_2]_{\approx} = c_0$; classe di equivalenza di $q_9 = [q_9]_{\approx} = c_3$;

AUTOMA QUOZIENTE

RICORDANDO PRIMA CHE: dato l'insieme A e definita la relazione di equivalenza '~' su A, l'insieme delle classi di equivalenza si chiama insieme quoziente di A rispetto alla relazione d'equivalenza ~ e si indica con A/~; Analogamente dati A e R (dove A è l'insieme e R una relazione di equivalenza di A) si dice insieme quoziente di A mod R l'insieme A/R, ossia l'insieme delle classi di equivalenza degli elementi A.

l'automa quoziente è un sistema $A_{\approx} = <\Sigma, Q/_{\approx}, \delta_{\approx}, [q_0]_{\approx}, F_{\approx}>$, dove:

- Σ, alfabeto sul quale l'automa lavora, prende anche il nome di "alfabeto di input", è uguale all'alfabeto di A;
- $Q/_{\approx}$, insieme delle classi di equivalenza di cui l'automa quoziente $A_{\approx}$, se Q è l'insieme degli stati dell'automa A, $Q/_{\approx}$ è l'insieme delle classi di equivalenza di Q costruite sulla relazione di indistinguibilità (≈);
- $\delta_{\approx}$, funzione di transizione che deve essere descritta:
  - mediante l'indicazione del dominio e codominio, ossia: $\delta_{\approx}: Q/_{\approx} \times \Sigma \rightarrow Q/_{\approx}$;
  - solo nel caso in cui sia il dominio che il codominio sono insiemi finiti allora $\delta_{\approx}$ è rappresentabile mediante tabella, dove nelle colonne vengono indicati i $q_i \in Q$ (con i che va da 0 ad h) e in riga vengono indicati i $\sigma_j \in \Sigma$ (con j che va da 1 a k), all'intersezione tra q_i e σ_j viene inserito la classe di equivalenza a cui appartiene lo stato che viene raggiunto dall'automa che si trova nello stato q_i e legge il simbolo σ_j

$\delta_{\approx}$	q_0	q_1	q_2	…	q_i	…	q_h
σ_1	$\delta_{\approx}([q_0]_{\approx}, \sigma_1)$	.	.	…	.	…	.
σ_2	.	.	$[\delta(q_2, \sigma_2)]_{\approx}$	…	.	…	.
σ_3	.	.	.	…	.	…	.
…	…	…	…	…	…	…	…
σ_j	.	.	.	…	$\delta_{\approx}([q_i]_{\approx}, \sigma_j)$	…	.
…	…	…	…	…	…	…	…
σ_k	.	$[\delta(q_1, \sigma_k)]_{\approx}$					

dove $[\delta(q, \sigma)]_{\approx} = \delta_{\approx}([q]_{\approx}, \sigma)$

- $[q_0]_{\approx}$, indica lo stato iniziale dell'automa quoziente che è definito dalla classe di equivalenza che contiene q_0, dove q_0 è lo stato iniziale dell'automa A;
- $F_{\approx}$, $F_{\approx} = \{[q]_{\approx} \mid q \in F\}$è l'insieme di tutte le classi di equivalenza che contengono stati finali (stati appartenenti all'insieme F dell'automa A), ossia quelle classi di equivalenza che, se raggiunte, mi consentono di ottenere uscita = 1;

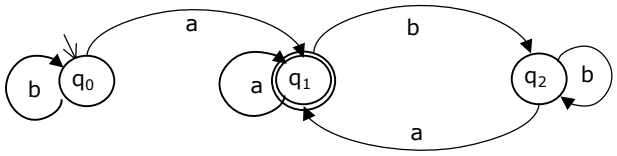
RICORDANDO PRIMA CHE: Per definizione due stati q_1 e q_2 si dicono indistinguibili e si scrive $q_1 \approx q_2$ se e solo se $\forall w \in \Sigma^*$ si ha che $\lambda(\delta^*(q_1, w)) = \lambda(\delta^*(q_2, w))$

può $[q]_{\approx}$ contenere almeno uno stato qj non finale se q è finale? NO, per appartenere alla stessa classe di equivalenza, gli stati devono essere tutti tra loro indistinguibili, quindi è fondamentale che siano tutti dello stesso tipo, o tutti finali o tutti non finali.

ESERCIZIO:

dato il seguente automa:

A = <Σ = {a, b}, Q = {q_0, q_1, q_2}, δ:leggibile dal disegno, q_0, F = {q_1}>



determinare:

1) il numero di stati dell'automa:
2) A è un automa osservabile?
3) A è un automa completamente specificato?
4) ∃ stati ∈ Q indistinguibili?
5) Definire $A_{\approx}$

1) numero di stati dell'automa

   si tratta di un automa a 3 stati q_0, q_1, q_2

2) A è un automa osservabile?

   si tratta di un automa osservabile in quanto, ogni stato, è osservabile, dove uno stato q si definisce osservabile quando esiste una parola w ∈ Σ* che permette di raggiungere lo stato q partendo dallo stato q_0, infatti:

   q_0 osservabile ↔ ∃ ε ∈ Σ* | δ*(q_0, ε) = q_0

   q_1 osservabile ↔ ∃ a ∈ Σ* | δ*(q_0, a) = q_1

   q_2 osservabile ↔ ∃ ab ∈ Σ* | δ*(q_0, ab) = q_2

3) A è un automa completamente specificato?

   si tratta di un automa completamente specificato in quanto, per ogni stato, esistono tanti archi uscenti quanti sono i simboli appartenenti a Σ, infatti:

   q_0 completamente specificato ↔ q_0 possiede 2 archi: arco etichettato con "a" e arco etichettato con "b"

   q_1 completamente specificato ↔ q_1 possiede 2 archi: arco etichettato con "a" e arco etichettato con "b"

   q_2 completamente specificato ↔ q_2 possiede 2 archi: arco etichettato con "a" e arco etichettato con "b"

4) ∃ stato ∈ Q indistinguibili?

   verifichiamo l'esistenza di stati indistinguibili tra loro:

   $q_0 \approx q_1$? NO, sono due stati distinguibili e questo è visibile già analizzando il disegno e osservando che il primo è uno stato non finale a differenza del secondo che risulta essere uno stato finale, queste due diverse tipologie di stato si distinguono sempre, infatti

   ∃ ε ∈ Σ | λ (δ*(q_0, ε)) = 0 ≠ λ(δ*(q_1, ε)) = 1

   $q_0 \not\approx q_1$

   $q_0 \approx q_2$? sono due stati dello stesso tipo, ossia non finali quindi devo trovare un ragionamento che mi permetta di negare l'indistinguibilità:

   cosa succede nei due stati se leggo ε? λ (δ*(q_0, ε)) = 0 = λ(δ*(q_2, ε)) = 0

   cosa succede nei due stati se leggo una sequenza di a? λ (δ*(q_0, a*)) = 1 = λ(δ*(q_2, a*)) = 1

   cosa succede nei due stati se leggo una sequenza di b? λ (δ*(q_0, b*)) = 0 = λ(δ*(q_2, b*)) = 0

   cosa succede nei due stati se leggo una sequenza mista di simboli? Finchè si legge una sequenza di b l'automa resta nello stato in cui si trova, non appena viene letta per la prima volta il carattere a i due stati convergono entrambi in q_0 e da quel momento sussistono con lo stesso percorso.

Si può quindi asserire che i due comportamenti sono analoghi quindi

$q_0 \approx q_2$

$q_1 \approx q_2$? NO, sono due stati distinguibili e questo è visibile già analizzando il disegno e osservando che il primo è uno stato non finale a differenza del secondo che risulta essere uno stato finale, queste due diverse tipologie di stato si distinguono sempre, infatti

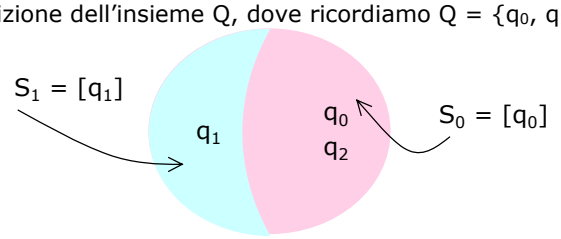
$\exists\ \varepsilon \in \Sigma \mid \lambda\ (\ \delta^*(q_1, \varepsilon)) = 1 \neq \lambda(\ \delta^*(q_2, \varepsilon)) = 0$

$q_1 \not\approx q_2$

5) definire $A_\approx$

sappiamo che: $q_0 \not\approx q_1$, $q_1 \not\approx q_2$ ma $q_0 \approx q_2$

effettuiamo la seguente partizione dell'insieme Q, dove ricordiamo $Q = \{q_0, q_1, q_2\}$



definiamo ora $A_\approx$:

$A_\approx = <\Sigma, Q/_\approx, \delta_\approx, [q_0]_\approx, F_\approx>$, dove:

$\Sigma = \{a, b\}$

$Q/_\approx = \{S_0, S_1\}$

$\delta_\approx, Q/_\approx \times \Sigma \rightarrow Q/_\approx$;

$\delta_\approx$	S_0	S_1
a	$\delta_\approx(S_0,a)$	$\delta_\approx(S_1,a)$
b	$\delta_\approx(S_0,b)$	$\delta_\approx(S_1,b)$

cosa significa effettuare $\delta_\approx(S_0,a)$? significa prendere un qualsiasi stato della classe di equivalenza S_0 e operarla con il simbolo "a", una volta ottenuto lo stato risultante dalla transizione, il risultato sarà la classe di equivalenza al quale appartiene lo stato in cui siamo arrivati.

$\delta_\approx(S_0,a) = \delta_\approx([q_0]_\approx,a) = [\delta(q_0,a)]_\approx = [q_1]_\approx = S_1$

$\delta_\approx(S_1,a) = \delta_\approx([q_1]_\approx,a) = [\delta(q_1,a)]_\approx = [q_1]_\approx = S_1$

$\delta_\approx(S_0,b) = \delta_\approx([q_0]_\approx,b) = [\delta(q_0,b)]_\approx = [q_0]_\approx = S_0$

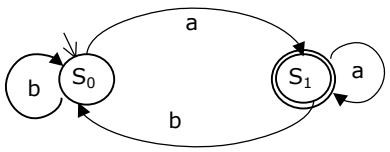
$\delta_\approx(S_1,b) = \delta_\approx([q_1]_\approx,b) = [\delta(q_1,b)]_\approx = [q_2]_\approx = S_0$

$\delta_\approx$	S_0	S_1
a	S_1	S_1
b	S_0	S_0

$[q_0]_\approx = S_0$

$F_\approx = \{[q]_\approx \mid q \in F\} = \{S_1\}$

$A_\approx = <\Sigma=\{a, b\}, Q/_\approx=\{S_0, S_1\}, \delta_\approx, S_0, F_\approx=\{S_1\}>$, dove:



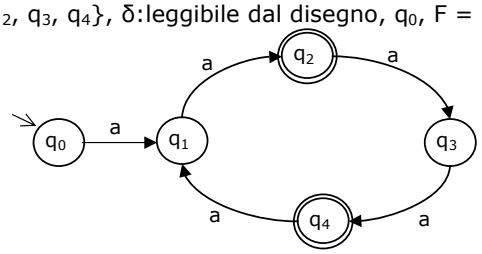
$L(A_\approx) = \{w \in \{a, b\}^* \mid \text{suffisso } w = \text{'a'}\} = \{a, b\}^*a$

$A_\approx$ è un'automa equivalente ad A, con il vantaggio che $A_\approx$ contiene uno stato in meno rispetto ad A.

ESERCIZIO:

dato il seguente automa:

A = <Σ = {a }, Q = {q_0, q_1, q_2, q_3, q_4}, δ:leggibile dal disegno, q_0, F = {q_2,q_4}>



determinare:

1) il numero di stati dell'automa:
2) A è un'automa osservabile?
3) A è un'automa completamente specificato?
4) ∃ stati ∈ Q indistinguibili?
5) Definire $A_{\approx}$

1) si tratta di un automa a 5 stati q_0, q_1, q_2, q_3, q_4
2) si tratta di un automa osservabile in quanto, ogni stato, è osservabile, dove uno stato q si definisce osservabile quando esiste una parola w ∈ Σ* che permette di raggiungere lo stato q partendo dallo stato q_0, infatti:
   q_0 osservabile ↔ ∃ ε ∈ Σ* | δ*(q_0, ε) = q_0
   q_1 osservabile ↔ ∃ a ∈ Σ* | δ*(q_0, a) = q_1
   q_2 osservabile ↔ ∃ a^2 ∈ Σ* | δ*(q_0, a^2) = q_2
   q_3 osservabile ↔ ∃ a^3 ∈ Σ* | δ*(q_0, a^3) = q_3
   q_4 osservabile ↔ ∃ a^4 ∈ Σ* | δ*(q_0, a^4) = q_4
3) si tratta di un automa completamente specificato in quanto, per ogni stato, esistono tanti archi uscenti quanti sono i simboli appartenenti a Σ, infatti:
   q_0 completamente specificato ↔ q_0 possiede l'arco etichettato con "a";
   q_1 completamente specificato ↔ q_1 possiede l'arco etichettato con "a";
   q_2 completamente specificato ↔ q_2 possiede l'arco etichettato con "a";
   q_3 completamente specificato ↔ q_3 possiede l'arco etichettato con "a";
   q_4 completamente specificato ↔ q_4 possiede l'arco etichettato con "a";
4) verifichiamo l'esistenza di stati indistinguibili tra loro:
   $q_0 \approx q_1$? cosa succede nei due stati se leggo "a"? λ (δ(q_0, a)) = 0 ≠ λ(δ(q_1, a)) = 1
   ho quindi ottenuto una parola w ∈ Σ* che mi differenzia il comportamento, quindi i due stati sono distinguibili
   $q_0 \not\approx q_1$
   $q_0 \approx q_2$? NO, sono due stati distinguibili dato che il primo è non finale e il secondo è finale, queste due diverse tipologie di stato si distinguono sempre per ε, infatti
   ∃ ε ∈ Σ | λ (δ(q_0, ε)) = 0 ≠ λ(δ(q_2, ε)) = 1
   $q_0 \not\approx q_2$
   $q_0 \approx q_3$? cosa succede nei due stati se leggo "a"? λ (δ(q_0, a)) = 0 ≠ λ(δ(q_3, a)) = 1
   ho quindi ottenuto una parola w ∈ Σ* che mi differenzia il comportamento, quindi i due stati sono distinguibili
   $q_0 \not\approx q_3$
   $q_0 \approx q_4$? NO, sono due stati distinguibili dato che il primo è non finale e il secondo è finale, queste due diverse tipologie di stato si distinguono sempre per ε, infatti
   ∃ ε ∈ Σ | λ (δ(q_0, ε)) = 0 ≠ λ(δ(q_4, ε)) = 1
   $q_0 \not\approx q_4$
   $q_1 \approx q_2$? NO, sono distinguibili in quanto il primo è non finale e il secondo è finale, si distinguono per ε,
   ∃ ε ∈ Σ | λ (δ(q_1, ε)) = 0 ≠ λ(δ(q_2, ε)) = 1
   $q_1 \not\approx q_2$
   $q_1 \approx q_3$? sono due stati dello stesso tipo, ossia non finali quindi devo trovare un ragionamento che mi permetta di negare l'indistinguibilità:
   cosa succede nei due stati se leggo ε? λ (δ(q_1, ε)) = 0 = λ(δ(q_3, ε)) = 0

cosa succede nei due stati se leggo una sequenza dispari di "a", ossia cosa succede se leggo a^{2n+1}?
$\lambda(\delta^*(q_1, a^{2n+1})) = 1 = \lambda(\delta^*(q_3, a^{2n+1})) = 1$
cosa succede nei due stati se leggo una sequenza pari di "a", ossia cosa succede se leggo a^{2n}?
$\lambda(\delta^*(q_1, a^{2n})) = 0 = \lambda(\delta^*(q_3, a^{2n})) = 0$
Si può quindi asserire che i due comportamenti sono analoghi quindi

$q_1 \approx q_3$

$q_1 \approx q_4$? NO, sono distinguibili in quanto il primo è non finale e il secondo è finale, si distinguono per ε,
$\exists\, \varepsilon \in \Sigma \mid \lambda(\delta(q_1, \varepsilon)) = 0 \neq \lambda(\delta(q_4, \varepsilon)) = 1$

$q_1 \not\approx q_4$

$q_2 \approx q_3$? NO, sono distinguibili in quanto il primo è finale e il secondo è non finale, si distinguono per ε,
$\exists\, \varepsilon \in \Sigma \mid \lambda(\delta(q_2, \varepsilon)) = 1 \neq \lambda(\delta(q_3, \varepsilon)) = 0$

$q_2 \not\approx q_3$

$q_2 \approx q_4$? sono due stati dello stesso tipo, ossia non finali quindi devo trovare un ragionamento che mi permetta di negare l'indistinguibilità:
cosa succede nei due stati se leggo ε? $\lambda(\delta(q_2, \varepsilon)) = 1 = \lambda(\delta(q_4, \varepsilon)) = 1$
cosa succede nei due stati se leggo una sequenza dispari di "a", ossia cosa succede se leggo a^{2n+1}?
$\lambda(\delta^*(q_2, a^{2n+1})) = 0 = \lambda(\delta^*(q_4, a^{2n+1})) = 0$
cosa succede nei due stati se leggo una sequenza pari di "a", ossia cosa succede se leggo a^{2n}?
$\lambda(\delta^*(q_2, a^{2n})) = 1 = \lambda(\delta^*(q_4, a^{2n})) = 1$
Si può quindi asserire che i due comportamenti sono analoghi quindi

$q_2 \approx q_4$

$q_3 \approx q_4$? NO, sono distinguibili in quanto il primo è non finale e il secondo è finale, si distinguono per ε,
$\exists\, \varepsilon \in \Sigma \mid \lambda(\delta(q_3, \varepsilon)) = 0 \neq \lambda(\delta(q_4, \varepsilon)) = 1$

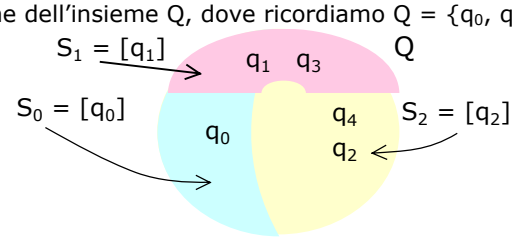
$q_3 \not\approx q_4$

5) determiniamo l'automa quoziente, $A_\approx$

sappiamo che: $q_0 \not\approx q_1$, $q_0 \not\approx q_2$, $q_0 \not\approx q_3$, $q_0 \not\approx q_4$, $q_1 \not\approx q_2$, $q_1 \not\approx q_4$, $q_2 \not\approx q_3$, $q_3 \not\approx q_4$

sappiamo che esistono le seguenti indistinguibilità: $q_1 \approx q_3$, $q_2 \approx q_4$

effettuiamo la seguente partizione dell'insieme Q, dove ricordiamo $Q = \{q_0, q_1, q_2, q_3, q_4\}$



definiamo ora $A_\approx$:

$A_\approx = <\Sigma, Q/_\approx, \delta_\approx, [q_0]_\approx, F_\approx>$, dove:

$\Sigma = \{a\}$

$Q/_\approx = \{S_0, S_1, S_2\}$

$\delta_\approx, Q/_\approx \times \Sigma \rightarrow Q/_\approx$;

$\delta_\approx$	S_0	S_1	S_2
a	$\delta_\approx(S_0,a)$	$\delta_\approx(S_1,a)$	$\delta_\approx(S_2,a)$

$\delta_\approx(S_0,a) = \delta_\approx([q_0]_\approx,a) = [\delta(q_0,a)]_\approx = [q_1]_\approx = S_1$

$\delta_\approx(S_1,a) = \delta_\approx([q_1]_\approx,a) = [\delta(q_1,a)]_\approx = [q_2]_\approx = S_2$

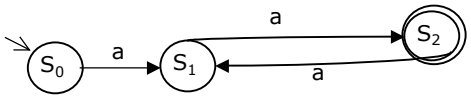
$\delta_\approx(S_2,a) = \delta_\approx([q_4]_\approx,a) = [\delta(q_4,a)]_\approx = [q_1]_\approx = S_1$

$\delta_\approx$	S_0	S_1	S_2
a	S_1	S_2	S_1

$[q_0]_\approx = S_0$

$F_\approx = \{[q]_\approx \mid q \in F\} = \{S_2\}$

$A_\approx = <\Sigma=\{a\}, Q/_\approx=\{S_0, S_1, S_2\}, \delta_\approx, S_0, F_\approx=\{S_2\}>$, dove:



$L(A_\approx) = \{w \in \{a\}^* \mid$ 'a' appaia nella parola un numero pari di volte$\} = a^{2n}$

$A_\approx$ è un'automa equivalente ad A, con il vantaggio che $A_\approx$ contiene due stati in meno rispetto ad A.

SINTESI OTTIMALE DI AUTOMI

Dato un linguaggio $L \subseteq \Sigma^*$, la sintesi ottimale di automi è quell'automa minimo per L, m_L, ossia l'automa più piccolo in grado di riconoscere L, ottenuto mediante la cancellazione degli stati indistinguibili dell'automa massimo per L, M_L, ossia l'automa osservabile più grande in grado di riconoscere il linguaggio in oggetto.

ALCUNE NOTAZIONI

w, indica la parola $w \in \Sigma^*$

$<w> \approx [w]$, indica lo stato che raggiunge l'automa leggendo la parola w partendo da q_0, $<w>=\delta(q_0, w)$

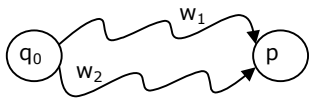
L'AUTOMA MASSIMO M_L

RICORDANDO PRIMA CHE: un automa A associa ad ogni $w \in \Sigma^*$ lo stato $\delta(q_0, w)$, definendo quindi una funzione $f: \Sigma^* \to Q$ dove $f(w) = \delta(q_0, w)$

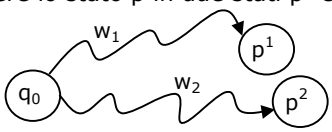
RICORDANDO PRIMA CHE: in un automa osservabile, per ogni stato q esiste una parola $w \in \Sigma^*$ tale che $q = \delta(q_0, w)$, ricordiamo che questa proprietà definisce la funzione come suriettiva.

Dato un linguaggio L, trovare l'automa massimo M_L significa trovare l'automa osservabile con il maggior numero di stati, ossia un automa ad infiniti stati, che definisca il linguaggio L dove parole diverse corrispondono a stati diversi, ossia se $w_1 \neq w_2$ deve esere $\delta(q_0, w_1) \neq \delta(q_0, w_2)$, questa proprietà definisce la funzione quale bijettiva.

Ipotizziamo di avere due parole w_1 e w_2 che si comportano nel seguente modo:



trovare l'automa massimo M_L significa scindere lo stato p in due stati p^1 e p^2, dove $p \approx p^1 \approx p^2$, ossia



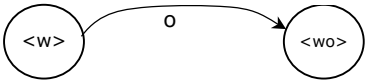
l'automa massimo ML è quindi un'automa dove, ciascuna parola deve raggiungere un diverso stato in modo da aumentare il numero di stati presenti dell'automa stesso e ciascuno stato deve essere raggiunto da una e una sola parola. Gli stati dell'automa massimo quindi sono definiti dall'insieme Σ^*.

L'automa massimo è quindi così definito:

- $\forall\, q \in Q\ \exists!\ w \in \Sigma^* \mid \delta^*(q_0, w) = q$
- $\forall\, w \in \Sigma^*\ \exists\ q \in Q \mid \delta^*(q_0, w) = q$

$M_L = <\Sigma, Q, \delta_M, <\varepsilon>, F>$, dove

- Σ, alfabeto sul quale l'automa massimo lavora;
- $Q = \{<w> \mid w \in \Sigma^*\}$ si osservi che gli stati di M_L sono essenzialmente le parole in Σ^*, quindi per qualsiasi L l'automa M_L ha sempre infiniti stati;
- δ_M, funzione di transizione che deve essere descritta:
  - mediante l'indicazione del dominio e codominio, ossia: $\delta_M: Q \times \Sigma \to Q$;
  - $\delta_M(<w>, \sigma) = <w\sigma>$



- $q_0 = <\varepsilon>$, lo stato iniziale è indicato dallo stato della parola vuota (ε);
- $F = \{<w> \in Q \mid w \in L\}$ dove L è il linguaggio che l'automa massimo riconosce;

OSSERVAZIONE:

siamo <x> e <y> due stati indistinguibili di M_L: i due stati si dicono indistinguibili se, leggendo una qualsiasi parola $w \in \Sigma^*$ il comportamento analogo sia partendo dallo stato <x> che partendo dallo stato <y>, ossia le uscite relative agli stati in cui arrivo dopo le due letture sono analoghe, quindi $\lambda(\delta_M(<x>, w)) = \lambda(\delta_M(<y>, w))$, sappiamo che $\delta_M(<k>, \sigma) = <k\sigma>$ quindi si deduce che devono essere indistinguibili le uscite degli stati derivanti dalla transizione, $\lambda(<xw>) = \lambda(<yw>)$, ma quando le uscite di due stati definiti secondo la dicitura $<k\sigma>$ sono uguali? Dato che $<k\sigma>$ individua lo stato raggiunto dall'automa partendo dallo stato <k> e leggendo il simbolo, o la parola σ, possiamo dedurre che $<k\sigma>$ ha uscita = 1 se e solo se $k\sigma$ è una parola appartenente al linguaggio, se $k\sigma$ non appartiene al linguaggio allora l'uscita sarà zero, è quindi possibile asserire che $<x> \approx <y>$ solo se $\forall\, w \in \Sigma^*$ se $xw \in L$ deve succedere che anche $yw \in L$, viceversa se $xw \notin L$ allora anche $yw \notin L$, questo per qualsiasi parola letta, riassumendo:

$<x> \approx <y> \leftrightarrow \forall\, w \in \Sigma^*\ \lambda(\delta_M(<x>, w)) = \lambda(\delta_M(<y>, w))$

$\leftrightarrow \forall\, w \in \Sigma^*\ \lambda(<xw>) = \lambda(<yw>)$

$\leftrightarrow \forall\, w \in \Sigma^*$ se $xw \in L \Rightarrow yw \in L$

OSSERVAZIONE:

siamo <k> e <j> due stati distinguibili di M_L: i due stati si dicono distinguibili se esiste almeno una parola w appartenente a Σ^* tale che il comportamento dell'automa, partendo dallo stato <k> e leggendo w sia differente dal comportamento dell'automa partendo dallo stato <j> e leggendo la medesima parola w, ossia le uscite relative agli stati in cui arrivo dopo le due letture sono distinte, quindi $\lambda(\delta_M(<k>, w)) \neq \lambda(\delta_M(<j>, w))$, sappiamo che $\delta_M(<k>, w) = <kw>$ quindi si deduce che devono essere distinguibili le uscite degli stati derivanti dalla transizione, $\lambda(<kw>) \neq \lambda(<jw>)$, ma quando le uscite di due stati definiti secondo la dicitura $<k\sigma>$ sono diverse? Dato che $<k\sigma>$ individua lo stato raggiunto dall'automa partendo dallo stato <k> e leggendo il simbolo, o la parola σ, possiamo dedurre che $<k\sigma>$ ha uscita = 1 se e solo se $k\sigma$ è una parola appartenente al linguaggio, se $k\sigma$ non appartiene al linguaggio allora l'uscita sarà zero, è quindi possibile asserire che $<k> \not\approx <j>$ solo se $\exists$ almeno una parola $w \in \Sigma^*$ | se $kw \in L$ deve succedere che $jw \notin L$, viceversa se $kw \notin L$ allora $jw \in L$, riassumendo:

$<k> \not\approx <j> \leftrightarrow \exists\, w \in \Sigma^* \ \lambda(\delta_M(<k>, w)) \neq \lambda(\delta_M(<j>, w))$

$\leftrightarrow \exists\, w \in \Sigma^* \ \lambda(<kw>) \neq \lambda(<jw>)$

$\leftrightarrow \exists\, w \in \Sigma^*$ se $kw \in L \Rightarrow jw \notin L$ oppure se $kw \notin L \Rightarrow jw \in L$

L'AUTOMA MINIMO m_L

Dato un linguaggio L, trovare l'automa minimo m_L significa trovare l'automa con il minor numero di stati, ossia l'automa più piccolo, che definisca il linguaggio L.

RICORDANDO PRIMA CHE: un automa si dice osservabile se, ogni stato, è osservabile;

RICORDANDO PRIMA CHE: uno stato q si dice osservabile se, esiste una parola $w \in \Sigma^*$ che permette di raggiungere lo stato q partendo dallo stato q_0, ossia: q osservabile $\leftrightarrow \exists\, w \in \Sigma^* \mid \delta^*(q_0, w) = q$

RICORDANDO PRIMA CHE: due stati q_1 e q_2 si dicono distinguibili e si scrive $q_1 \not\approx q_2$ se e solo se esiste una parola che distingue il comportamento in uscita dell'automa, ossia: $\exists\, w \in \Sigma^* | \lambda\,(\delta^*(q_1, w)) \neq \lambda(\delta^*(q_2, w))$

PROPRIETÀ DI m_L:

- Tutti gli stati di m_L devono essere osservabili, quindi m_L deve essere un'automa osservabile;
- Tutti gli stati di m_L devono essere distinguibili tra loro, in caso contrario, infatti, avrei degli stati cancellabili e quindi l'automa preso in considerazione non sarebbe l'automa minimo;
- Non può esistere un automa A per L dove A ha un numero di stati minore di m_L, ossia $\#stati(A) \geq \#stati(m_L)$ quest'ultima affermazione è dimostrata per assurdo qui di seguito:

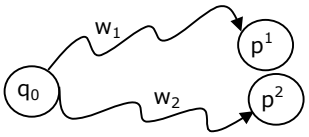
DIMOSTRAZIONE PER ASSURDO:

supponiamo di avere un automa $A = <\Sigma, Q, q_0, \delta, F>$ che riconosce il linguaggio L tale che il numero di stato di A sia inferiore del numero di stati di m_L, dove ricordiamo che con la dicitura m_L intendiamo l'automa minimo per il linguaggio L.

ciò significa che troviamo due parole w_1 e w_2 che, in A raggiungono il medesimo stato, mentre in m_L raggiungono due stati distinti, ossia

A

m_L

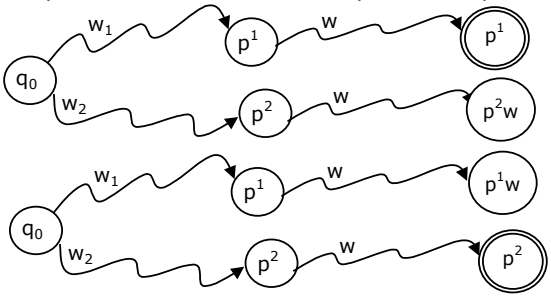


$\exists\, w_1, w_2 \mid$

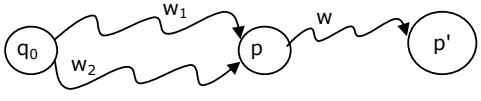
dato che p^1 e p^2 sono due stati dell'automa minimo mL allora deve valere che: p^1 è uno stato distinguibile da p^2 per definizione di automa minimo, ossia $p^1 \not\approx p^2$, ma come detto in precedenza:

$<p^1> \not\approx <p^2> \leftrightarrow \exists\, w \in \Sigma^*$ se $p^1w \in L \Rightarrow p^2w \notin L$ oppure se $p^1w \notin L \Rightarrow p^2w \in L$

m_L



A



in A: p' o è finale o non è finale ovvero:

se p' è finale $\Rightarrow \lambda(\delta^*(p, w)) = 1 \Rightarrow \lambda\,(\delta^*(<w_1>, w)) = \lambda\,(\delta^*(<w_2>, w)) = 1 \Rightarrow w_1w, w_2w \in L$

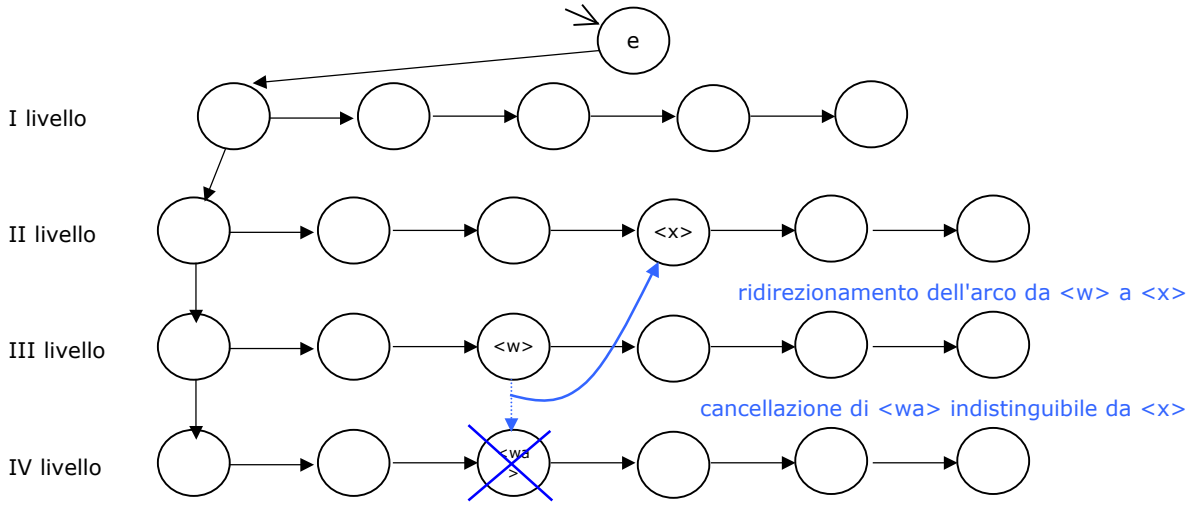
se p' non è finale $\Rightarrow \lambda(\delta^*(p, w)) = 0 \Rightarrow \lambda\,(\delta^*(<w_1>, w)) = \lambda\,(\delta^*(<w_2>, w)) = 0 \Rightarrow w_1w, w_2w \notin L$

ma questo è in contrasto con quanto definito da m_L (ASSURDO), quindi A non è in grado di definire in maniera corretta il linguaggio L definito dall'automa minimo mL, infatti il linguaggio mL accetta una sola tra le due parole w_1w e w_2w, l'automa A invece o accetta entrambe le parole oppure le rifiuta entrambe, se ne deduce che i due automi riconoscono due linguaggi differenti.

ALGORITMO PER COSTRUIRE m_L (AUTOMA MINIMO) PARTENDO DA M_L (AUTOMA MASSIMO):

- trovare l'automa massimo M_L per il linguaggio L, dove M_L viene descritto da un albero con infiniti nodi;
- partendo dalla radice <ε>, visito M_L in ampiezza (per livelli) effettuando a ciascun nodo il "confronto del nodo corrente <wσ> con i nodi già visitati", ossia:
  - parto da <ε>
  - passo al livello 1
    - esamino il primo nodo (<primo>) del livello 1 con <ε> per verificare se è indistinguibile da <ε>;
    - esamino il secondo nodo (<secondo>) del livello 1 prima con <ε> e poi con <primo> per verificare se è indistinguibile da uno degli stati già esaminati;
    - esamino il terzo nodo (<terzo>) del livello 1 prima con <ε> se risulta distinguibile da <ε> esamino <terzo> con <primo> e infine con <secondo>, sempre per verificare se è indistinguibile da uno degli stati già esaminati;
    - proseguo con tutti gli stati del livello 1;
  - passo al livello 2
    - esamino il primo nodo (<primo2>) del livello 2 con <ε>, <primo>, <secondo>, <terzo>, <…> ossia con <ε> e tutti i nodi del livello 1;
    - esamino il secondo nodo (<secondo2>) del livello 2 con <ε>, <primo>, <secondo>, <terzo>, <…>, <primo2> ossia con <ε> , tutti i nodi del livello 1 e il primo nodo del livello 2;
    - esamino il terzo nodo (<terzo2>) del livello 2 con <ε>, <primo>, <secondo>, <terzo>, <…>, <primo2> e <secondo2> ossia con <ε> , tutti i nodi del livello 1 e i primi due nodi del livello 2;
    - proseguo con tutti gli stati del livello 2;
  - passo al livello 3 e così via per tutti i livelli presenti nell'automa;
- evidenzio gli stati indistinguibili <wσ> trovati nei vari livelli effettuando il "confronto dei nodi" dell'automa massimo
- ipotizziamo per esempio che, visitando il nodo <wσ> del livello 4, cui arriva un arco etichettato con σ dal nodo <w> del livello 3, abbiamo trovato indistinguibilità tra <wσ> è il nodo <x> del livello 2, già precedentemente visitato in quanto presente in un livello superiore rispetto al livello 4 in cui si trova il nodo indistinguibile, effettuo la seguente operazione:
  - cancello <wσ>
  - cancello l'intero sottoalbero derivante dal nodo <wσ>
  - ricordando che <wσ> è figlio dello stato <w> mediante l'utilizzo dell'arco σ, cosa succede all'arco che consentiva dallo stato <w> di passare a <wσ> leggendo "σ"? l'arco viene ridirezionato verso lo stato <x> con il quale <wσi> è stato confrontato durante il "confronto del nodo corrente <wσ> con il nodo già visitato <x>" quando il risultato del confronto ha evidenziato l'indistinguibilità tra <x> e <w.
- Non è sempre possibile ottenere un'automa a stati finiti, alcuni linguaggi infatti accettano automi minimi, ma detti automi minimi sono comunque automi a sfati non finiti.

Nel diagramma sottostante è stato rappresentato un esempio di "confronto di nodi correnti con i nodi già visitati", è stato rilevata un'indistinguibilità tra lo stato <x> appartenente al livello II e lo stato <wa> che viene raggiunto dall'arco "a" che parte dallo stato <w> appartenente al livello III, il diagramma rappresenta con una x azzurra la cancellazione dello stato <wa> ≈ <x> e il riorientamento dell'arco da <w> ad <x>



ESEMPIO DI LINGUAGGIO CHE ACCETTA AUTOMA A STATI FINITI

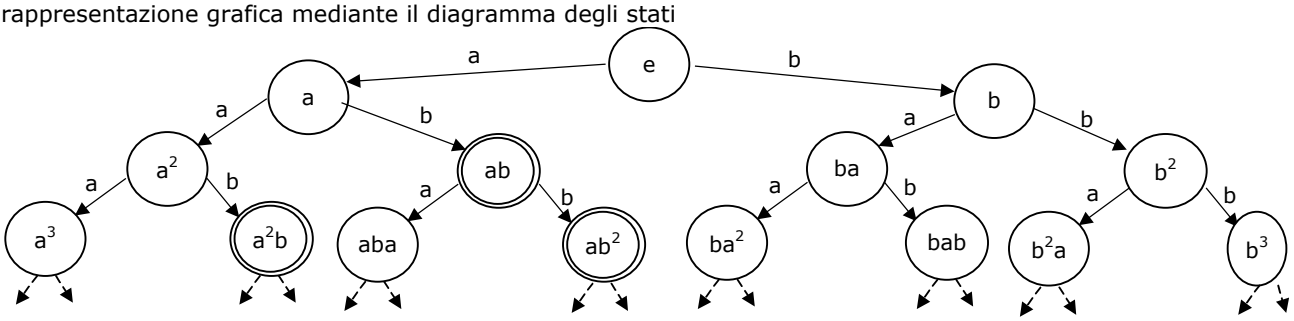
dato il linguaggio $L = \{a^n b^m \mid n, m > 0\} = a^+b^+$

costruzione degli stati appartenenti a Q dell'automa massimo M_L:

stato iniziale <ε>	ε ∉ L	⇒	<ε> non finale
δ_M (<ε>, a) = <a>	a ∉ L	⇒	<a> non finale
δ_M (<ε>, b) = <b>	b ∉ L	⇒	<b> non finale
δ_M (<a>, a) = <aa> = <a^2>	a^2 ∉ L	⇒	<a^2> non finale
δ_M (<a>, b) = <ab>	ab ∈ L	⇒	<ab> FINALE
δ_M (<b>, a) = <ba> = <ba>	ba ∉ L	⇒	<ba> non finale
δ_M (<b>, b) = <bb> = <b^2>	b^2 ∉ L	⇒	<b^2> non finale
δ_M (<a^2>, a) = <a^2a> = <a^3>	a^3 ∉ L	⇒	<a^3> non finale
δ_M (<a^2>, b) = <a^2b>	a^2b ∈ L	⇒	<a^2b> FINALE

...

rappresentazione grafica mediante il diagramma degli stati



controlliamo i diversi livelli per trovare gli stati indistinguibili:

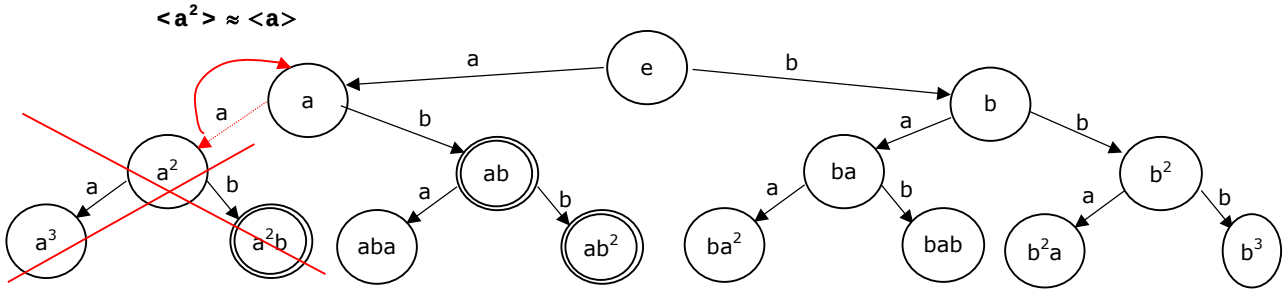
livello I:

<a> ≈ <ε>?	No, infatti ∃ "b" | λ(δ(<ε>, b)) = 0 ≠ λ(δ(<a>, b)) = 1	⇒ **<a> ≉ <ε>**
<b> ≈ <ε>?	No, infatti ∃ "ab" | λ(δ(<ε>, ab)) = 1 ≠ λ(δ(<b>, ab)) = 0	⇒ **<b> ≉ <ε>**
<b> ≈ <a>?	No, infatti ∃ "b" | λ(δ(<a>, b)) = 1 ≠ λ(δ(<b>, b)) = 0	⇒ **<b> ≉ <a>**

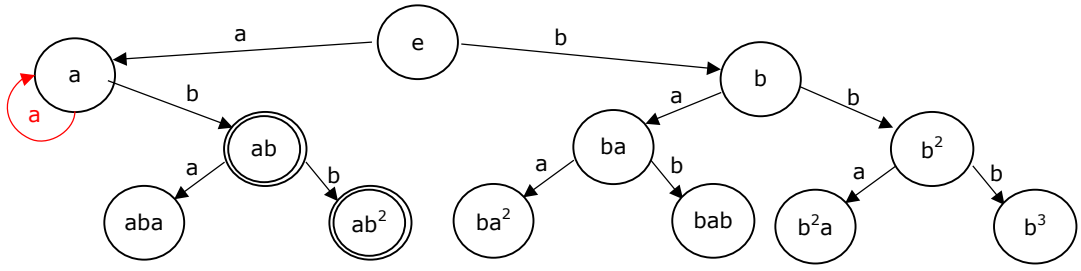
livello II:

<a^2> ≈ <ε>?	No, infatti ∃ "b" | λ(δ(<ε>, b)) = 0 ≠ λ(δ(<a^2>, b)) = 1	⇒ **<a^2> ≉ <ε>**
<ab> ≈ <ε>?	No, infatti <ab> è finale mentre <ε> è uno stato non finale	⇒ **<ab> ≉ <ε>**
<ba> ≈ <ε>?	No, infatti ∃ "ab" | λ(δ(<ε>, ab)) = 1 ≠ λ(δ(<ba>, ab)) = 0	⇒ **<ba> ≉ <ε>**
<b^2> ≈ <ε>?	No, infatti ∃ "ab" | λ(δ(<ε>, ab)) = 1 ≠ λ(δ(<b^2>, ab)) = 0	⇒ **<b^2> ≉ <ε>**
<a^2> ≈ <a>?	SI, sono indistinguibili infatti ∀ w ∈ Σ* l'automa si comporta analogamente sia partendo da <a> che partendo da <a^2>, quindi bisogna effettuare la modifica, eliminare <a^2>, eliminare il suo sottoalbero, ridirezionare l'arco che porta in <a^2>	

<a^2> ≈ <a>



ottenendo quindi il seguente diagramma degli stati:



<ab> ≈ <a>? No, infatti <ab> è finale mentre <a> è uno stato non finale ⇒ **<ab> ≉ <a>**

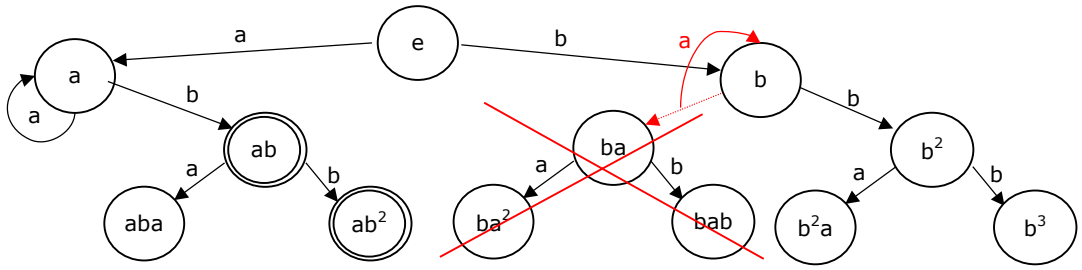
<ba> ≈ <a>? No, infatti ∃ "b" | λ(δ(<a>, b)) = 1 ≠ λ(δ(<ba>, b)) = 0 ⇒ **<ba> ≉ <a>**

<b^2> ≈ <a>? No, infatti ∃ "b" | λ(δ(<a>, b)) = 1 ≠ λ(δ(<b^2>, b)) = 0 ⇒ **<b^2> ≉ <a>**

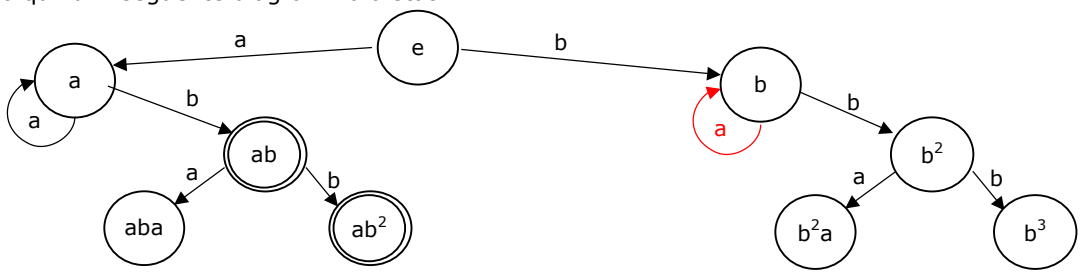
<ab> ≈ <b>? No, infatti <ab> è finale mentre <b> è uno stato non finale ⇒ **<ab> ≉ <b>**

<ba> ≈ <b>? SI, sono indistinguibili infatti ∀ w ∈ Σ* l'automa si comporta analogamente sia partendo da <b> che partendo da <ba>, infatti qualsiasi parole venga letta non otterrò mai stati di tipo finale, quindi bisogna effettuare la modifica, eliminare <ba>, eliminare il suo sottoalbero, ridirezionare l'arco che porta in <ba>

<ba> ≈ <b>

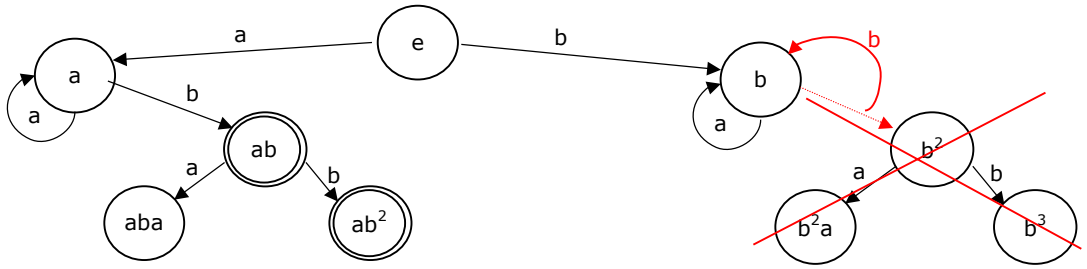


ottenendo quindi il seguente diagramma a stati:

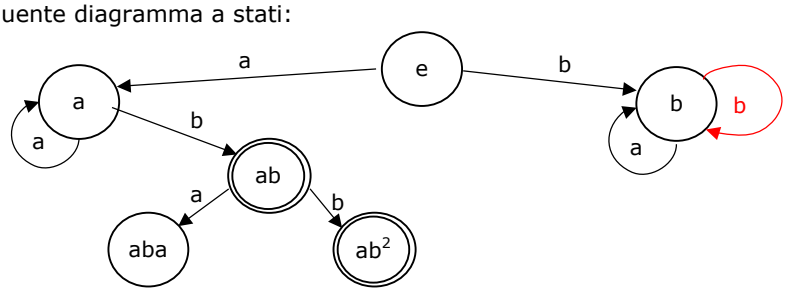


<b^2> ≈ <b>? SI, sono indistinguibili infatti ∀ w ∈ Σ* l'automa si comporta analogamente sia partendo da <b> che partendo da <b^2>, infatti qualsiasi parole venga letta non otterrò mai stati di tipo finale, quindi bisogna effettuare la modifica, eliminare <b^2>, eliminare il suo sottoalbero, ridirezionare l'arco che porta in <b^2>

<b^2> ≈ <b>



ottenendo quindi il seguente diagramma a stati:



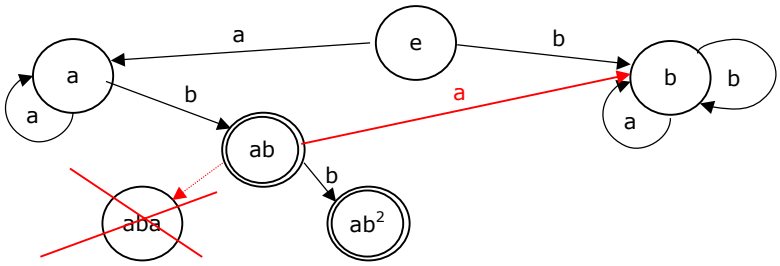
livello III:

<aba> ≈ <ε>? No, infatti ∃ "ab" | λ(δ(<ε>, ab)) = 1 ≠ λ(δ(<aba>, ab)) = 0 ⇒ **<aba> ≉ <ε>**

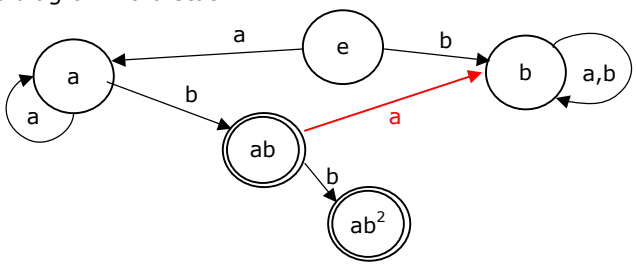
<aba> ≈ <a>? No, infatti ∃ "b" | λ(δ(<a>, b)) = 1 ≠ λ(δ(<aba>, b)) = 0 ⇒ **<aba> ≉ <a>**

<aba> ≈ <b>? SI, sono indistinguibili infatti ∀ w ∈ Σ* l'automa si comporta analogamente sia partendo da <b> che partendo da <aba>, infatti qualsiasi parole venga letta non otterrò mai stati di tipo finale, quindi bisogna effettuare la modifica, eliminare <aba>, eliminare il suo sottoalbero, ridirezionare l'arco che porta in <aba>

<aba> ≈ <b>



ottenendo quindi il seguente diagramma a stati:



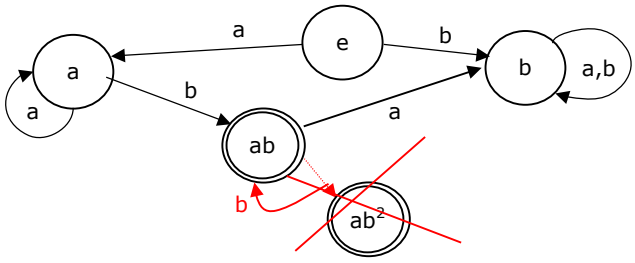
<ab²> ≈ <ε>? No, infatti <ab²> è finale mentre <ε> è uno stato non finale ⇒ **<ab²> ≉ <ε>**

<ab²> ≈ <a>? No, infatti <ab²> è finale mentre <a> è uno stato non finale ⇒ **<ab²> ≉ <a>**

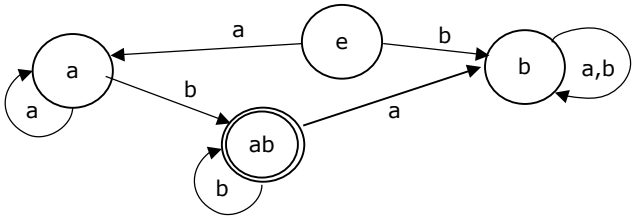
<ab²> ≈ <b>? No, infatti <ab²> è finale mentre <b> è uno stato non finale ⇒ **<ab²> ≉ <b>**

<ab²> ≈ <ab>? SI, sono indistinguibili infatti entrambi non appena leggono una "a" non possono più fornire risultato = 1, finchè leggono "b" possono fornire risultato uguale ad 1. ∀ w ∈ Σ* l'automa si comporta analogamente sia partendo da <ab²> che partendo da <ab>, quindi bisogna effettuare la modifica, eliminare <ab²>, eliminare il suo sottoalbero, ridirezionare l'arco che porta in <ab²>

<ab²> ≈ <ab>



ottenendo quindi il seguente diagramma degli stati:



questo è lo schema finale

ESEMPIO DI LINGUAGGIO CHE NON ACCETTA AUTOMA A STATI FINITI

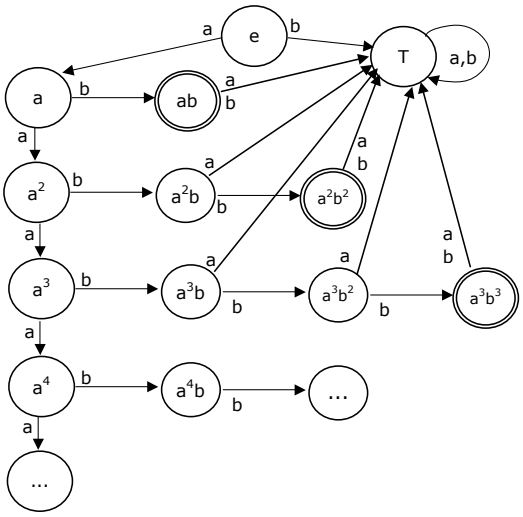
dato il linguaggio $L = \{a^n b^n \mid n > 0\}$

costruzione degli stati appartenenti a Q dell'automa massimo M_L:

stato iniziale $<\varepsilon>$	$\varepsilon \notin L$	$\Rightarrow$	$<\varepsilon>$ non finale
$\delta_M(<\varepsilon>, a) = <a>$	$a \notin L$	$\Rightarrow$	$<a>$ non finale
$\delta_M(<\varepsilon>, b) = <T>$	$b \notin L$	$\Rightarrow$	$<T>$ stato trappola
$\delta_M(<a>, a) = <aa> = <a^2>$	$a^2 \notin L$	$\Rightarrow$	$<a^2>$ non finale
$\delta_M(<a>, b) = <ab>$	$ab \in L$	$\Rightarrow$	$<ab>$ FINALE

...

rappresentazione grafica mediante il diagramma degli stati



controlliamo i diversi livelli per trovare gli stati indistinguibili:

$<a> \approx <\varepsilon>$? No, infatti $\exists$ "b" | $\lambda(\delta(<\varepsilon>, b)) = 0 \neq \lambda(\delta(<a>, b)) = 1$ $\Rightarrow$ **$<a> \not\approx <\varepsilon>$**

$<a^2> \approx <\varepsilon>$? No, infatti $\exists$ "b^2" $|\lambda(\delta(<\varepsilon>, b^2)) = 0 \neq \lambda(\delta(<a^2>, b^2)) = 1$ $\Rightarrow$ **$<a^2> \not\approx <\varepsilon>$**

$<ab> \approx <\varepsilon>$? No, infatti $<ab>$ è finale mentre $<\varepsilon>$ è uno stato non finale $\Rightarrow$ **$<ab> \not\approx <\varepsilon>$**

$<a^2> \approx <a>$? No, infatti $\exists$ "b" $|\lambda(\delta(<a>, b)) = 1 \neq \lambda(\delta(<a^2>, b)) = 0$ $\Rightarrow$ **$<a^2> \not\approx <a>$**

IMPORTANTE, SI NOTI CHE: $\forall\ a^i, a^j$ con $i \neq j$ si ha che gli stati $<a^i>$ e $<a^j>$ sono SEMPRE tra loro distinguibili, infatti $\exists\ b^i \mid \lambda(\delta(<a^i>, b^i)) = 1 \neq \lambda(\delta(<a^j>, b^i)) = 0$

$<ab> \approx <a>$? No, infatti $<ab>$ è finale mentre $<a>$ è uno stato non finale $\Rightarrow$ **$<ab> \not\approx <a>$**

$<a^3> \approx <\varepsilon>$? No, infatti $\exists$ "b^3" $|\lambda(\delta(<\varepsilon>, b^3)) = 0 \neq \lambda(\delta(<a^3>, b^3)) = 1$ $\Rightarrow$ **$<a^3> \not\approx <\varepsilon>$**

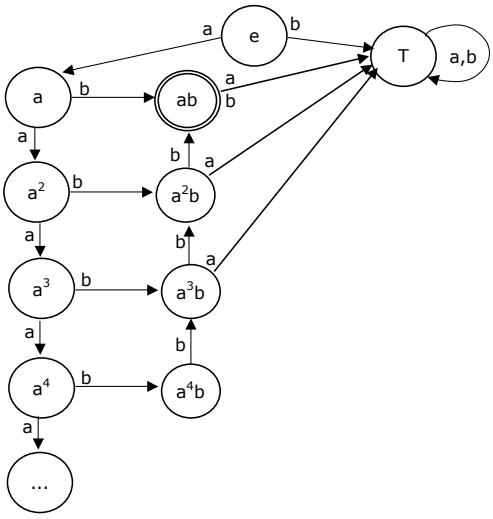
$<a^2b> \approx <\varepsilon>$? No, infatti $\exists$ "b" | $\lambda(\delta(<\varepsilon>, b)) = 0 \neq \lambda(\delta(<a^2b>, b)) = 1$ $\Rightarrow$ **$<a^2b> \not\approx <\varepsilon>$**

$<a^3> \approx <a>$? No, infatti $\exists$ "b" | $\lambda(\delta(<a>, b)) = 1 \neq \lambda(\delta(<a^3>, b)) = 0$ $\Rightarrow$ **$<a^3> \not\approx <a>$**

$<a^2b> \approx <a>$? No, infatti $\exists$ "ab^2" | $\lambda(\delta(<a>, ab^2)) = 1 \neq \lambda(\delta(<a^2b>, ab^2)) = 0$ $\Rightarrow$ **$<a^2b> \not\approx <a>$**

dallo studio dei vari livelli si evinceranno le seguenti indistinguibilità:

- $<ab> \approx <a^2b^2> \approx <a^3b^3> \approx <a^4b^4> \approx <a^5b^5> \approx <a^xb^x>$, ossia l'indistinguibilità di tutti gli stati finali
- tra gli stati del tipo $<a^ib>$ con gli stati del tipo $<a^{i-1}b>$



SE L È LINGUAGGIO REGOLARE ↔ ∃ AUTOMA A STATI FINITI CHE LO DEFINISCE

RICORDANDO PRIMA CHE: si parla di linguaggio regolare quando un linguaggio appartiene alla classe dei linguaggi R_3

RICORDANDO PRIMA CHE: si parla di linguaggio appartenente alla classe dei linguaggi R_3 se ammette una grammatica G di tipo 3 che lo genera

TEOREMA:

in questa sezione proviamo che i linguaggi regolari (di tipo 3) sono esattamente quelli riconosciuti da automi a stati finiti; gli automi a stati finiti risultano dunque i riconoscitori corrispondenti a quei sistemi generativi che sono le grammatiche di tipo 3; Il teorema di equivalenza dice che:

1) se un linguaggio è un linguaggio regolare ⇒ esiste un automa a stati finiti che lo definisce
2) se un automa a stati finiti riconosce un linguaggio ⇒ questo è regolare.

Il teorema quindi fornisce un'equivalenza tra le grammatiche di tipo 3 e gli automi a stati finiti. Per asserire che quest'equivalenza esiste dobbiamo dimostrare i due punti del teorema di equivalenza.

DIMOSTRAZIONE:

2) L RICONOSCIUTO DA DFA ⇒ ∃ G DI TIPO 3

sia L il linguaggio riconosciuto dall'automa a stati finiti $A = <\Sigma^A, Q^A, q_0{}^A, \delta^A, F^A>$, possiamo costruire la grammatica $G = <\Sigma^G, M^G, S^G, P^G>$ di tipo 3 procedendo nel seguente modo:

- $\Sigma^G = \Sigma^A$ l'alfabeto dei simboli terminali non cambia rispetto all'alfabeto dei simboli terminali dell'automa
- $M^G = Q^A$ i metasimboli della grammatica sono gli stati dell'automa
- $S^G = q_0{}^A$ l'assioma di partenza per la grammatica è lo stato iniziale dell'automa
- P^G:
  - $q \to \varepsilon$ se q è uno stato finale nell'automa A, dove $q \in Q = M$, quindi dove q è metasimbolo per G
  - $q \to \sigma p$ dove p è lo stato raggiunto da q seguente l'arco etichettato con σ, ossia $q \to \sigma p \leftrightarrow \delta(q, \sigma) = p$

si noti bene che la generazione dell'insieme delle regole di produzione segue le caratteristiche richieste perchè si possa parlare di grammatica di tipo 3.

CORRETTEZZA DI G

RICORDANDO PRIMA CHE: si dice che R è una relazione d'ordine sull'insieme X se soddisfa le proprietà di riflessività, transitività e antisimmetria.

Le regole di produzione di tipo 3 sono definibili quali relazioni d'ordine infatti

- riflessiva: $\forall q, qp \in Q, \forall p \in P$ si ha che $q \Rightarrow^p qp$
- transitiva: $\forall q, r, s \in Q, \forall p \in P$ se $q \Rightarrow^p r$ e $r \Rightarrow^p s \Rightarrow q \Rightarrow^* s$
- antisimmetria e definita dal fatto che le relazioni di tipo 3 sono relazioni in cui non possono essere effettuate cancellazioni di caratteri di derivazione.

RICORDANDO PRIMA CHE: il principio di induzione prima forma si applica tutte le volte che si vuole dimostrare una proprietà P(n) che dipende da un insieme sul quale esiste una relazione d'ordine, preso l'insieme X, il principio d'induzione si effettua come segue:

- devo trovare un n_0 tale che $p(n_0)$ sia vara
- suppongo di riuscire a dimostrare che P(n-1) sia vera allora devo indurre P(n) vera
- se riesco a far vedere che i punti soprastanti sono verificati allora sono sicura che P(n) è vera per tutti gli n.

Si necessita di mostrare che G gode della seguente proprietà: $\forall w \in \Sigma^* \mid \delta(q, w) = p \leftrightarrow$ nella grammatica che sto costruendo accade che da q posso derivare, in uno o più passi, la forma sentenziale wp, ossia,

$\forall w \in \Sigma^* \; \delta(q, w) = p \leftrightarrow q \Rightarrow_G{}^* wp$

per dimostrare che questo principio è vero necessitiamo del principio di induzione nella prima forma, seguendo le direttive sopra citate nel "ricordando prima che" procediamo con la dimostrazione di $P(n_0)$ vera, per poi verificare anche il punto 2) del principio:

- $P(n_0)$: n_0 significa che $|w| = 0$, ossia che la lunghezza della parola w è zero ($w = \varepsilon$)

Dimostriamo che: $\delta(q, \varepsilon) = q \leftrightarrow q \Rightarrow^* \varepsilon \cdot q = q$

Questa proposizione risulta vera in quanto la derivazione in zero passi non effettua transizioni di stato da parte dell'automa, analogamente nella grammatica significa effettuare la concatenazione tra lo stato e il simbolo ε

- P(n-1) vera ⇒ P(n) vera, per ipotesi di induzione sappiamo che la proprietà è vera per uno stato x con $|x|=n-1$, si necessita di dimostrare che la proprietà è vera anche per uno stato con $|x| = n$, detto stato è per esempio xσ, infatti $|x\sigma| = |x| + |\sigma| = (n-1) + 1 = n$

Dimostriamo che: $\delta(q, x\sigma) = p \leftrightarrow q \Rightarrow^* x\sigma p$

Cos'è $\delta(q, x\sigma)$? $\delta(q, x\sigma) = \delta(\delta(q, x), \sigma)$ dove $\delta(q, x)$ rappresenta lo stato che raggiungo partendo da q e leggendo x, questo sarà lo stato dal quale partirò nel cammino per leggere l'ultimo simbolo σ.

Dato che per ipotesi di induzione $\delta(q, x)$ è vera ossia $\exists\ d \mid q \Rightarrow^* xq$ allora $\delta(\delta(q, x), \sigma) = p$ è una transizione dell'automa DFA, dato che la grammatica G è costruita in base all'automa avremo, nell'insieme P la seguente regola di produzione: $xq \rightarrow^{\sigma} p$

Dato che: $q \Rightarrow^* xq$ (vera per ipotesi), $xq \rightarrow^{\sigma} p$ (vera per costruzione di P) $\Rightarrow q \Rightarrow^* x\sigma p$

A questo punto l'automa dopo aver letto $x\sigma$ deve decidere se l'uscita (1/0?)

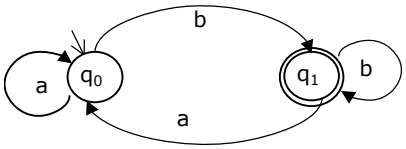
$\lambda(\delta(\delta(q, x), \sigma)=p) =$

- 1 se $p \in F$, e nella grammatica deve succedere che $q \Rightarrow^* x\sigma p$ e inoltre, dato che p deve essere finale deve esistere la regola di produzione $p \rightarrow \varepsilon$, in questo modo posso ottenere la produzione finale $q \Rightarrow^* xop \Rightarrow x\sigma$, ossia $q \Rightarrow^* x\sigma$
- 0 se $p \notin F$, e nella grammatica deve succedere che $q \Rightarrow^* x\sigma p$ e inoltre, dato che p deve essere NON finale deve mancare dall'insieme P delle regole di produzione della grammatica G, la regola di produzione $p \rightarrow \varepsilon$, in questo modo posso ottenere la produzione finale $q \Rightarrow^* xop$, la presenza di un metasimbolo (p) nella produzione mi definisce che non ho ottenuto una parola finale della grammatica G.

ESEMPIO DALL'AUTOMA ALLA GRAMMATICA

L = {a, b}*b

Ossia sia L il linguaggio che riconosce una sequenza di simboli {a, b} che devono necessariamente terminare con suffisso "b".



definiamo ora la grammatica G di tipo 3 per mezzo dell'automa A appena definito:

- $\Sigma^G = \Sigma^A = \{a, b\}$
- $M^G = Q^A = \{q_0, q_1\}$
- $S^G = q_0{}^A = q_0$
- P^G:
  1. $q_1 \rightarrow \varepsilon$ in quanto q_1 è finale
  2. $q_0 \rightarrow aq_0$
  3. $q_0 \rightarrow bq_1$
  4. $q_1 \rightarrow aq_0$
  5. $q_1 \rightarrow bq_1$

$G = <\Sigma = \{a, b\}, M = \{q_0, q_1\}, q_0, P = \{q_1 \rightarrow \varepsilon \mid aq_0 \mid bq_1, q_0 \rightarrow aq_0 \mid bq_1\}>$

Test di G :

1) "abb" $\in$ L(A)

verifichiamo se G è in grado di generare una parola riconosciuta dal linguaggio definito dall'automa A:

$q_0 \Rightarrow_G{}^2 aq_0 \Rightarrow_G{}^3 abq_1 \Rightarrow_G{}^5 abbq_1 \Rightarrow_G{}^1 abb$

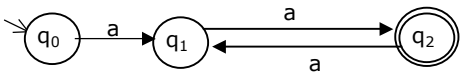
2) "abba" $\notin$ L(A)

$q_0 \Rightarrow_G{}^2 aq_0 \Rightarrow_G{}^3 abq_1 \Rightarrow_G{}^5 abbq_1 \Rightarrow_G{}^4 abbaq_0$

non è possibile cancellare q_0 in quanto non abbiamo la regola di produzione della forma $q_0 \rightarrow \varepsilon$ quindi la parola "abba" non risulta generabile dalla grammatica G.

ESEMPIO DALL'AUTOMA ALLA GRAMMATICA

Sia dato il seguente automa, descritto dal diagramma a stati sottostanti.



A = <Σ= {a}, Q={q_0, q_1, q_2}, q_0, δ definito da disegno, F={q_2}>

definiamo ora la grammatica G di tipo 3 per mezzo dell'automa A appena definito:

- $\Sigma^G = \Sigma^A = \{a, b\}$
- $M^G = Q^A = \{q_0, q_1, q_2\}$
- $S^G = q_0{}^A = q_0$
- P^G:
  1. $q_2 \rightarrow \varepsilon$ in quanto q_2 è finale
  2. $q_0 \rightarrow aq_1$
  3. $q_1 \rightarrow aq_2$
  4. $q_2 \rightarrow aq_1$

$G = <\Sigma = \{a\}, M = \{q_0, q_1, q_2\}, q_0, P = \{q_0 \rightarrow aq_1, q_1 \rightarrow aq_2, q_2 \rightarrow \varepsilon \mid aq_1\}>$

1) L REGOLARE ⇒ ∃ DFA (DETERMINISTIC FINAL AUTOMA)

RICORDANDO PRIMA CHE: Le grammatiche di tipo 3 sono identificate da regole di produzione che assumono una delle seguenti quattro tipologie di forme:

1) $A \to x$ oppure $A \to yB$ (dette lineari destre)
2) $A \to \sigma$ oppure $A \to \sigma B$ oppure $A \to \varepsilon$
3) $A \to \sigma B$ oppure $A \to \varepsilon$
4) $A \to \sigma B$ oppure $A \to \sigma$

dove: $A, B \in M$, $x, y \in \Sigma^*$, $\sigma \in \Sigma$

RICORDANDO PRIMA CHE: Queste quattro forme possono essere trasformate da una all'altra senza modificare il linguaggio che la grammatica genera

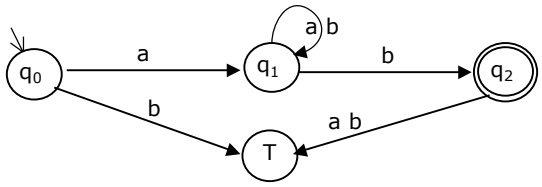
Sia L il linguaggi generato a partire dalla grammatica G di tipo 3 con regole di produzione nella forma 3) ossia $(A \to \sigma B | A \to \varepsilon)$, dove $G = <\Sigma^G, M^G, S^G, P^G>$, possiamo costruire l'automa $DFA = <\Sigma^A, Q^A, q_0{}^A, \delta^A, F^A>$ così definito:

- $\Sigma^A = \Sigma^G$ l'alfabeto dei simboli terminali non cambia rispetto a quello della grammatica
- $Q^A = M^G$ gli stati dell'automa sono i metasimboli della grammatica G
- $q_0{}^A = S^G$ lo stato iniziale dell'automa è determinato dall'assioma di partenza per la grammatica
- δ^A: $\exists\, q \to \sigma p \in P^G \Rightarrow \delta(q, \sigma) = p$ per ogni produzione del tipo $q \to \sigma p$ definisco la regola $\delta(q, \sigma) = p$ in A
- $F^A = \{A \in M^G \mid \exists\, A \to \varepsilon \text{ in } G\}$ è uno stato finale tutti quegli stati che in G hanno la regola $A \to \varepsilon$

ESEMPIO DALLA GRAMMATICA ALL'AUTOMA

L(G) = linguaggio costruito sulla grammatica G

$G = <\Sigma = \{a, b\}, M = \{q_0, q_1, q_2\}, q_0, P = \{ q_0 \to aq_1, q_1 \to aq_1 | bq_1 | bq_2, q_2 \to \varepsilon\}>$ definiamo ora l' automa A per mezzo della grammatica G appena definita:



interpretando i metasimboli come stati, osserviamo che il grafo precedente non è il grafo degli stati di un automa. in un automa, infatti, dato uno stato q e un simbolo σ esiste un unico stato prossimo $\delta(q, \sigma)$, nel grafo appena costruito invece, ci sono due transizioni etichettate con "b" che portano da q_1 a due distinti stati q_1 e q_2. possiamo interpretare questo fatto come una specie di non determinismo, se il sistema si trova in un dato stato, l'arrivo di un messaggio non porta necessariamente ad uno stato prossimo univocamente individuato dallo stato presente e dal messaggio, bensì porta in uno tra un insieme di stati prossimi possibili. si tratta di un automa NON DETERMINISTIC FINAL AUTOMA a causa della coesistenza della regola di produzione $q_1 \to bq_1 | bq_2$, $A_{NFA} = <\Sigma^A, Q^A, q_0{}^A, R^A, F^A>$

- $\Sigma^A = \Sigma^G = \{a, b\}$
- $Q^A = M^G = \{q_0, q_1, q_2\}$
- $q_0{}^A = S^G = q_0$
- R^A: $Q \times \Sigma \times Q \to \{0, 1\}$
  - $R(q_0, a, q_0) = 0$ non esiste la regola di produzione $q_0 \to aq_0$
  - $R(q_0, b, q_0) = 0$ non esiste la regola di produzione $q_0 \to bq_0$
  - $R(q_0, a, q_1) = 1$ $\exists\, q_0 \to aq_1 \in P$
  - $R(q_0, b, q_1) = 0$ non esiste la regola di produzione $q_0 \to bq_1$
  - $R(q_0, a, q_2) = 0$ non esiste la regola di produzione $q_0 \to aq_2$
  - $R(q_0, b, q_2) = 0$ non esiste la regola di produzione $q_0 \to bq_2$
  - $R(q_1, a, q_0) = 0$ non esiste la regola di produzione $q_1 \to aq_0$
  - $R(q_1, b, q_0) = 0$ non esiste la regola di produzione $q_1 \to bq_0$
  - $R(q_1, a, q_1) = 1$ $\exists\, q_1 \to aq_1 \in P$
  - $R(q_1, b, q_1) = 1$ $\exists\, q_1 \to bq_1 \in P$
  - $R(q_1, a, q_2) = 0$ non esiste la regola di produzione $q_1 \to aq_2$
  - $R(q_1, b, q_2) = 1$ $\exists\, q_1 \to bq_2 \in P$
  - $R(q_2, a, q_0) = 0$ non esiste la regola di produzione $q_2 \to aq_0$
  - $R(q_2, b, q_0) = 0$ non esiste la regola di produzione $q_2 \to bq_0$
  - $R(q_2, a, q_1) = 0$ non esiste la regola di produzione $q_2 \to aq_1$
  - $R(q_2, b, q_1) = 0$ non esiste la regola di produzione $q_2 \to bq_1$
  - $R(q_2, a, q_2) = 0$ non esiste la regola di produzione $q_2 \to aq_2$
  - $R(q_2, b, q_2) = 0$ non esiste la regola di produzione $q_2 \to bq_2$
- $F^A = \{q_2\}$

DFA E NFA

Gli automi si suddividono in due grosse tipologie a seconda se l'automa imponga o meno delle scelte durante le transizioni tra i diversi stati di cui l'automa stesso si compone:

- NFA, NOT DETERMINISTIC FINAL AUTOMA, sono quegli automi che impongono delle scelte durante la transizione tra gli stati di cui si compone l'automa stesso.
- DFA, DETERMINISTIC FINAL AUTOMA, sono quegli automi che non impongono delle scelte durante la transizione tra gli stati di cui si compone l'automa stesso, è un particolare caso di automa non deterministico.

NFA

Un automa a stati finiti non deterministico è un sistema A = <Σ, Q, q_0, R, F> dove Q è un insieme finito di stati, Σ è un alfabeto finito e R è l'insieme delle relazioni di transizione, non si parla quindi più di funzione (δ: Q x Σ → Q) bensì di relazione di transizione non determinastaca:

R: Q x Σ x Q → {0, 1}, dove R può essere definita:

- Elencando tutte le possibili relazioni di transizione tra i diversi stati utilizzando le regole di relazione;
- Elencando solo le relazioni di transizione tra i diversi stati che restituiscono 1, in questo caso le relazioni di transizione tra gli stati che restituiscono 0 sono sottointese;

una relazione di transizione viene definita nel seguente modo:

R (q, σ, p) = 0 indica che non è possibile effettuare la transizione dallo stato q allo stato p mediante la lettura di σ, ossia che non esiste un arco etichettato con σ tra lo stato q e lo stato p

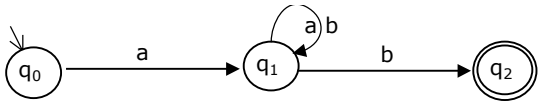
R (q, σ, p) = 1 indica che è possibile effettuare la transizione dallo stato q allo stato p mediante la lettura di σ, ossia che esiste un arco etichettato con σ tra lo stato q e lo stato p.

Qual è il linguaggio riconosciuto da un NFA? Una volta identificato l'automa dobbiamo determinare il tipo di linguaggio che l'automa definisce, che prenderà il nome di L(A_{NFA}) ed è l'insieme di tutte le parole riconosciute dall'automa. Mentre in un DFA una parola induce un solo cammino, in NFA una parola potrebbe indurre più cammini, infatti data in input una parola w all'automa A, dove A è un NFA, lo stato indotto dalla transizione può essere sia uno stato finale, sia uno stato non finale, sia lo stato trappola.

Una parola w ∈ Σ* si dice accettata da NFA se esiste almeno un cammino per w che termina in uno stato finale (dove ricordiamo che gli stati finali sono gli stati accettanti ossia gli stati che consentono uscita = 1), ossia,

w accettata ↔ {w ∈ Σ* | w induce almeno un cammino dallo stato iniziale ad uno stato ∈ F}

ESEMPIO DI NFA E DI PAROLE ACCETTATE



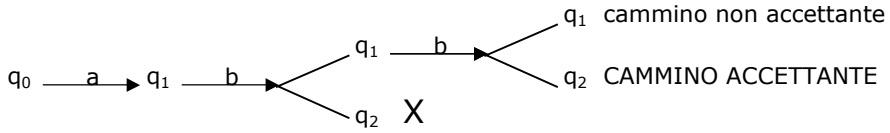
A_{NFA} = <Σ = {a, b}, Q = {q_0, q_1, q_2}, q_0, R: Q x Σ x Q → {0, 1}, F = {q_2}>

Dove R: Q x Σ x Q → {0, 1}

- o R(q_0, a, q_1) = 1
- o R(q_1, a, q_1) = 1
- o R(q_1, b, q_1) = 1
- o R(q_1, b, q_2) = 1

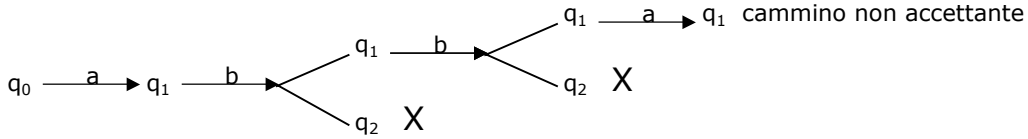
R è stata definita mediante l'elencazione delle transizioni con R(q, σ, p) = 1, i casi R(q, σ, p) = 0 sono sottointesi

Sia L(A_{NFA}) il linguaggi riconosciuto dall'automa A_{NFA}, definiamo se « abb » e « abba » sono parole appartenenti al linguaggio L(A_{NFA})



la "X" indica che l'automa si blocca e quindi sottointende un cammini non accettante

"abb" è accettata grazie all'esistenza del cammino q_0 → q_1 → q_1 → q_2, dove q_2 è finale quindi accettante



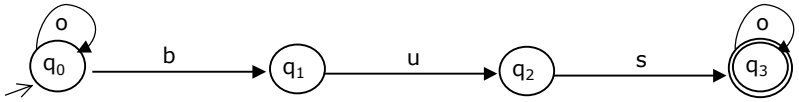
"abba" non è accettata in quanto non esiste neanche un cammino accettante.

Cos'è L(A_{NFA})? Si tratta di un linguaggio che necessita della lettura di una parola che inizia per "a", nel caso in cui, infatti, l'automa leggesse quale primo carattere della parola w il simbolo "b" si bloccherebbe immediatamente, e termini per "b", infatti la lettura del simbolo "b" induce nello stato q_2, dove l'automa resta se "b" è l'ultimo simbolo letto.

L (A_{NFA}) = {w ∈ {a, b}* | w = k·h·j con k = "a" e j = "b"} = a{a, b}*b

ESEMPIO DI NFA "LO STRING MATCHING"

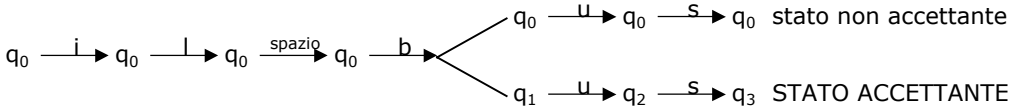
Vogliamo costruire un automa in grado di trovare la parola "bus" in un testo w, dove w è il messaggio fornito in input all'automa:



A è un automa non deterministico? Si, infatti sia lo stato q_0 induce in due diversi stati alla lettura del simbolo "b" infatti o transita in q_1 oppure transita in q_0.

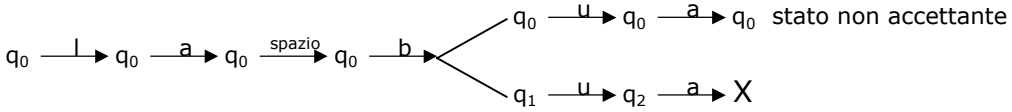
Proviamo a vedere come si comporta l'automa A nel riconoscere le seguenti parole:

"il bus":



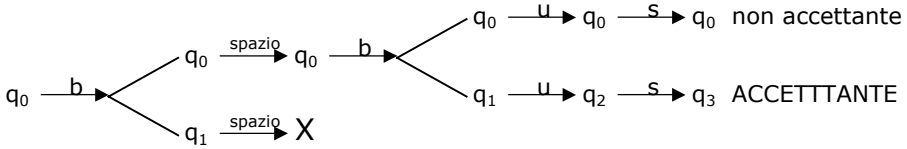
∃ un cammino accettante che termina nello stato finale q_3 quindi "il bus" è una parola riconosciuta dall'automa

"la bua":



in un caso la transizione termina in uno stato non accettante, nell'altro si blocca quindi la parola non è riconosciuta dall'automa, infatti, non esiste neanche un cammino accettante che termina nello stato finale q_3.

"b bus":



∃ un cammino accettante che termina nello stato finale q_3 quindi "b bus" è una parola riconosciuta dall'automa

DFA

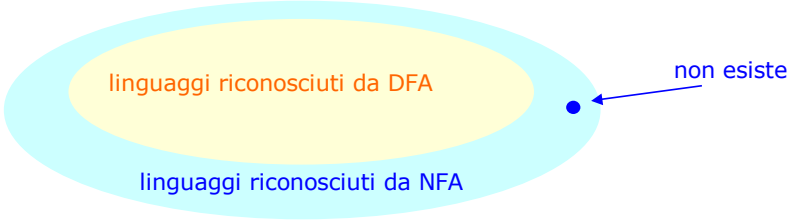
Un automa a stati finiti deterministico A = <Σ, Q, q_0, δ, F> è un particolare automa non deterministico A = <Σ, Q, q_0, R, F> dove è sempre possibile sostituire la funzione di transizione δ con una relazione di transizione R, infatti R(q, σ, p) = 1 ↔ δ(q, σ) = p.

$L_{DFA} \subseteq L_{NFA}$?

L_{NFA} = insieme dei linguaggi riconosciuti con automi a stati finiti non deterministici

L_{DFA} = insieme dei linguaggi riconosciuti con automi a stati finiti deterministici

È corretto asserire che $L_{DFA} \subseteq L_{NFA}$? Per poter provare che i linguaggi riconosciuti dagli automi deterministici sono un sottoinsieme dei linguaggi riconosciuti dai non deterministici bisognerebbe provare l'esistenza di un linguaggio riconosciuto da un automa a stati finiti non deterministico che non può essere riconosciuto con un automa equivalente ma deterministico, esiste detto linguaggio? No non esiste in quanto un qualsiasi linguaggi riconosciuto da un automa non deterministico è anche riconoscibile da un automa deterministico equivalente, si deduce quindi:



∀ NFA ∃ DFA EQUIVALENTE

RICORDANDO PRIMA CHE: dati due automi A e B questi si dicono equivalenti se riconoscono lo stesso linguaggio, ossia, A equivalente a B ↔ L(A) =L(B)

per ogni automa a stati finiti non deterministico (NFA) è possibile effettuare la costruzione di un automa a stati finiti deterministico (DFA) a lui equivalente.

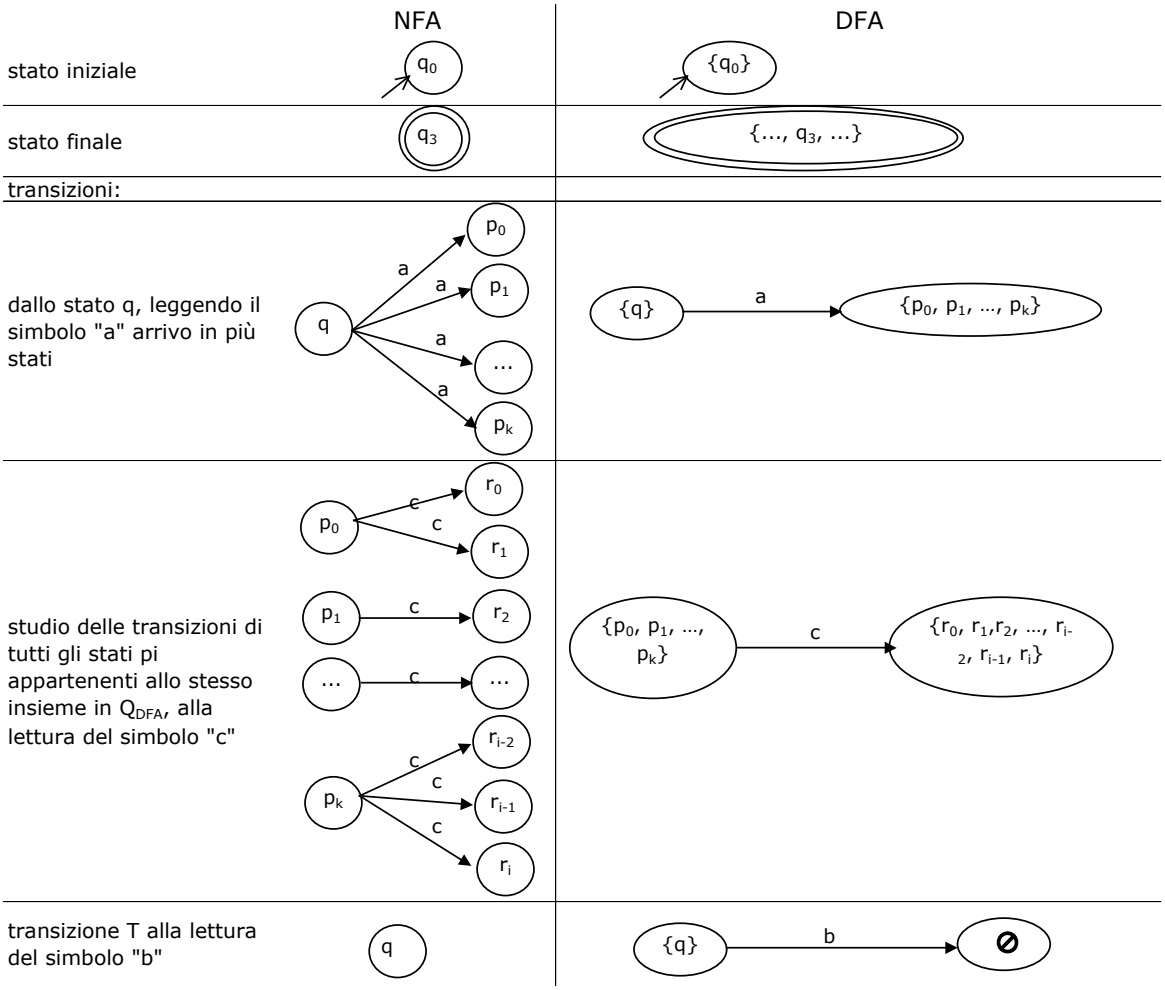
TEOREMA DI TRASFORMAZIONE

ALCUNE NOTAZIONI FONDAMENTALI

Q_0 indica lo stato iniziale, $q_0 \in Q_{NFA}$ — $\{Q_0\}$ indica l'insieme contenente lo stato q_0, dove $\{q_0\} \in Q_{DFA}$

T indica lo stato trappola di un NFA — $\varnothing$ indica l'insieme vuoto, ossia lo stato trappola di un DFA

ALGORITMO INFORMALE DI TRASFORMAZIONE



si noti che $\{q_1, q_2, \ldots, q_n\}$ è finale se $\exists$ almeno uno stato finale tra i diversi stati che compongono l'insieme.

ALGORITMO FORMALE DI TRASFORMAZIONE

Per ogni linguaggi L riconosciuto da un automa a stati finiti non deterministico, abbiamo asserito fin'ora che esiste anche un automa a stati finiti deterministico che lo riconosce.

Sia $A_{NFA} = <\Sigma_{NFA}, Q_{NFA}, R_{NFA}, q_{0NFA}, F_{NFA}>$ un automa a stati finiti non deterministico per il linguaggio L, vogliamo procedere alla costruzione di $A_{DFA} = <\Sigma_{DFA}, Q_{DFA}, \delta_{DFA}, q_{0DFA}, F_{DFA}>$ dove:

- $\Sigma_{DFA} = \Sigma_{NFA}$
- $Q_{DFA} = 2^{QNFA}$ gli stati del DFA sono un sottoinsieme degli stati dell'NFA, dove 2QNFA indica l'insieme dei sottoinsiemi dell'insieme Q_{NFA}, quanti sono i sottoinsiemi di Q_{NFA}? Ossia quanti sono gli elementi di Q_{DFA}? Sono pari a $|2^{QNFA}| = 2^{|QNFA|}$
  ESEMPIO: sia $Q_{NFA} = \{p, q\} \Rightarrow Q_{DFA} = 2^{QNFA} = \{\varnothing, \{p\}, \{q\}, \{p, q\}\}$
- δ_{DFA}: $Q_{DFA} \times \Sigma_{DFA} \rightarrow Q_{DFA}$ ossia δ_{DFA}: $2^{QNFA} \times \Sigma \rightarrow 2^{QNFA}$ tale che, se $S \in 2^{QNFA}$, ossia se S è uno stato dell'automa A_{DFA} e $\sigma \in \Sigma$, allora dobbiamo esprimere formalmente cos'è $\delta(S, \sigma)$? Supponiamo per ora che S contenga un solo stato, lo stato p, allora $\delta(S, \sigma)$ sono quegli stati $q \in Q_{NFA}$ tali che si verifichi la condizione $R(p, \sigma, q) = 1$ ossia partendo dallo stato p e leggendo il simbolo σ si arrivi nello stato q, quindi $\delta(S, \sigma) = \{q \in Q_{NFA} \mid R(p, \sigma, q) = 1\}$, non è detto però che S contenga solo lo stato p, S infatti, in quanto insieme potrebbe contenere più stati, quindi devo considerare p come una variabile e analizzare tutti quegli stati $p \in S$, si ottiene quindi la necessità di effettuare l'unione tra tutti quegli stati q raggiunti partendo dai diversi stati $p \in S$ e leggendo il simbolo σ, si ottiene quindi che $\delta(S, \sigma) = \cup \{q \in Q_{NFA} \mid R(p, \sigma, q) = 1\}$
- $q_{0DFA} = \{q_0\}$
- $F_{DFA} = \{Y \in Q_{DFA} \mid \exists\, y \in (Y \cap F_{NFA})\} = \{Y \in 2^{QNFA} \mid \exists\, y \in (Y \cap F_{NFA})\} = \{Y \in 2^{QNFA} \mid Y \cap F_{NFA} \neq \varnothing\}$, ossia cerco quegli stati tali che sia possibile trovare uno stato contenuto in Y che sia finale, dove per essere finale deve appartenere a F_{NFA}.

SVANTAGGI DELLA TRASFORMAZIONE

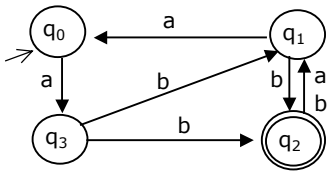
Sebbene un NFA è trasformabile in un DFA che riconosce lo stesso linguaggio, il DFA equivalente ha un numero di stati molto elevato rispetto al NFA

$$\#\text{ stati NFA} = k \Rightarrow k \leq \#\text{stati DFA} \leq 2^k$$

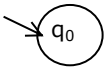
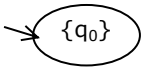
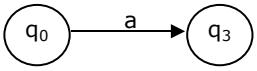
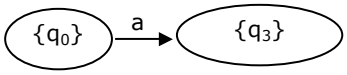
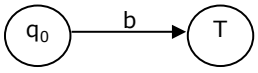
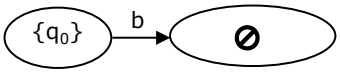
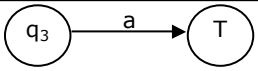
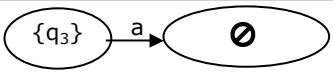
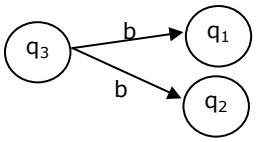
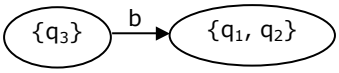
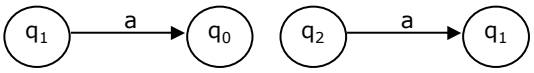
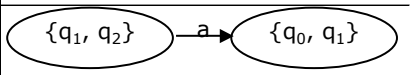
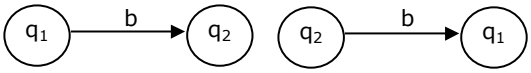
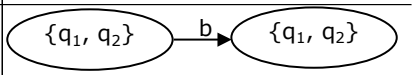
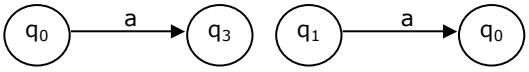
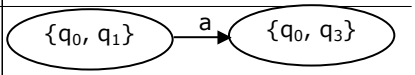
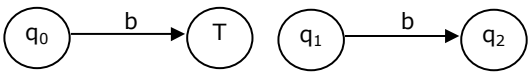
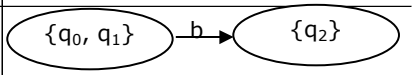
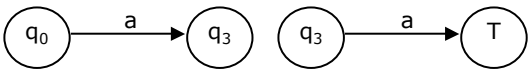
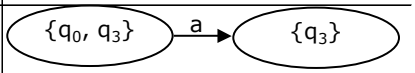
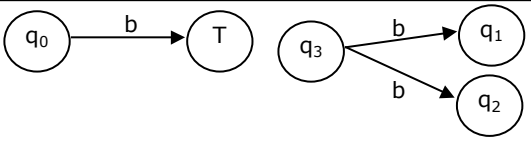
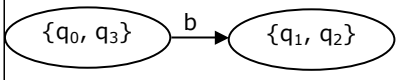
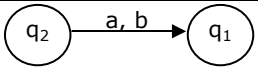
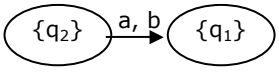
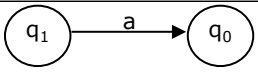
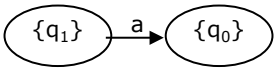
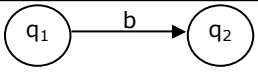
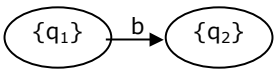
se il numero di stati dell'automa NFA è k allora il numero di stati del DFA equivalente è un valore che può variare tra k e 2^k, è quindi possibile avere, nel peggiore dei casi, un incremento esponenziale.

ESERCIZIO DA NFA A DFA

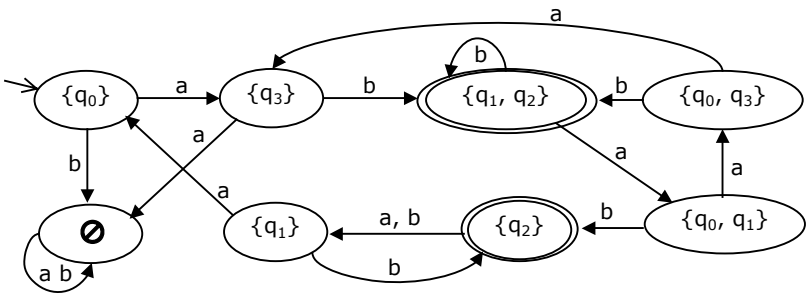
Preso il seguente automa NFA:



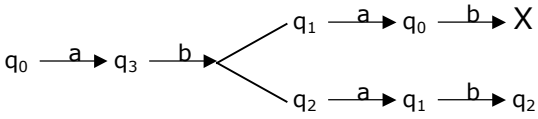
trasformiamolo in un DFA:

	NFA	DFA
stato iniziale		
transizioni:		
dallo stato q_0, leggendo "a"		
dallo stato q_0, leggendo "b"		
dallo stato q_3, leggendo "a"		
dallo stato q_3, leggendo "b"		
dagli stati q_1, q_2 leggendo "a"		
dagli stati q_1, q_2 leggendo "b"		
dagli stati q_0, q_1 leggendo "a"		
dagli stati q_0, q_1 leggendo "b"		
dagli stati q_0, q_3 leggendo "a"		
dagli stati q_0, q_3 leggendo "b"		
dallo stato q_2, leggendo a, b		
dallo stato q_1, leggendo "a"		
dallo stato q_1, leggendo "b"		

quali sono gli stati finali dell'automa DFA? Tutti quegli insiemi che contengono lo stato q_2, ossia $\{q_2\}$ e $\{q_1,q_2\}$ si ottiene quindi il seguente automa a stati finiti deterministico:



prendiamo la parola w_1 = "abab" e vediamo se è accettata dal NFA e dal DFA



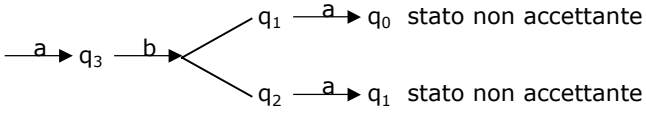
NFA:

la parola w_1 è accettata dal NFA in quanto ∃ un cammino che termina in uno stato accettante

DFA: $\{q_0\} \xrightarrow{a} \{q_3\} \xrightarrow{b} \{q_1, q_2\} \xrightarrow{a} \{q_0, q_1\} \xrightarrow{b} \{q_2\}$

dato che $q_2 \in Q_{NFA}$ è finale allora $\{q_2\} \in 2^{QNFA}$ è finale quindi la parola viene accettata

prendiamo la parola w_2 = "aba" e vediamo se è accettata dal NFA e dal DFA



NFA:

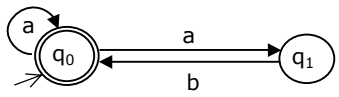
la parola w_2 non è accettata dal NFA in quanto non esiste neanche un cammino che termina in uno stato accettante

DFA: $\{q_0\} \xrightarrow{a} \{q_3\} \xrightarrow{b} \{q_1, q_2\} \xrightarrow{a} \{q_0, q_1\}$

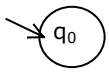
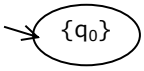
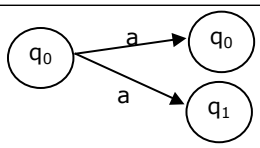
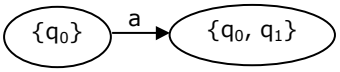
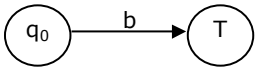
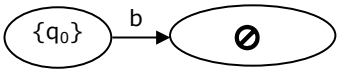
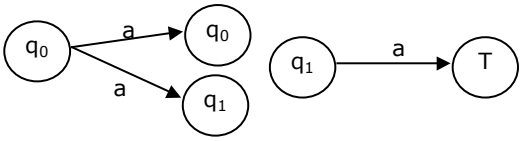
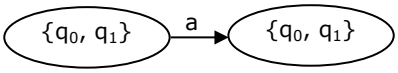
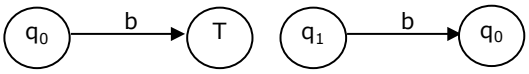
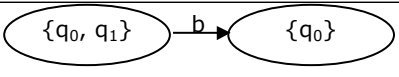
dato che $q_0, q_1 \in Q_{NFA}$ sono entrambi due stati non finali allora $\{q_0, q_1\} \in 2^{QNFA}$ è non finale quindi la parola non viene accettata

ESERCIZIO DA NFA A DFA

Preso il seguente automa NFA:

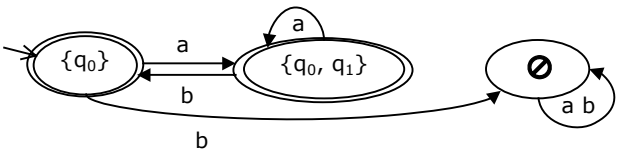


trasformiamolo in un DFA:

	NFA	DFA
stato iniziale		
transizioni:		
dallo stato q_0, leggendo "a"		
dallo stato q_0, leggendo "b"		
dagli stati q_0, q_1 leggendo "a"		
dagli stati q_0, q_1 leggendo "b"		

quali sono gli stati finali dell'automa DFA? Tutti quegli insiemi che contengono lo stato q_0, ossia $\{q_0,q_1\}$ e $\{q_0\}$

si ottiene quindi il seguente automa a stati finiti deterministico:



ESPRESSIONI REGOLARI

Le espressioni regolari (ER) servono per denotare un linguaggio, la definizione di espressione regolare è fatta in maniera induttiva, (analogamente a come sono state date le definizioni di espressioni booleane e espressioni aritmetiche a pagina 4).

Definizione di ER su Σ:

- Sono espressioni regolari base:
  - $\varnothing$
  - ε
  - $\sigma \in \Sigma$
- se p e q sono espressioni regolari allora
  - $p + q$ è espressione regolare
  - $p \cdot q$ è espressione regolare
  - p^* è espressione regolare, dove lo * indica la chiusura di kleene

abbiamo detto che le espressioni regolari (ER) servono per denotare un linguaggio L, qui di seguito ad ogni espressione regolare p associamo un linguaggio L (e diremo che p denota L) come segue:

ER	L	ER	L
$\varnothing$	$\varnothing$	q	B (linguaggio)
ε	$\{\varepsilon\}$	$p + q$	$A \cup B$
σ	$\{\sigma\}$	$p \cdot q$	$A \cdot B$
p	A (linguaggio)	p^*	A^*

osserviamo che le espressioni regolari denotano linguaggi in modo composizionale, una data espressione indica le operazioni di unione, prodotto e chiusura che, applicate ai linguaggi base $\varnothing$, $\{\varepsilon\}$, $\{\sigma\}$, permettono di ottenere il linguaggio denotato. Questo permette l'uso di tecniche induttive per mostrare proprietà di linguaggi denotati da espressioni regolari. Ad esempio, per mostrare che una proprietà P vale per tutti i linguaggi denotati da espressioni regolari è sufficiente provare che:

- i linguaggi base verificano la proprietà
- se i linguaggi A e B verificano la proprietà $\Rightarrow$ $A \cup B$, $A \cdot B$ e A^* verificano la proprietà

ESEMPIO DI ESPRESSIONE REGOLARE

$a^*b^*a^*+(ab)^*$ è un'espressione regolare che denota il linguaggio L dove:

$L = \{w \mid w = a^j b^k a^l$ (con j, k, l ≥ 0) oppure $w = (ab)^h$ (con h ≥ 0)$\}$

ESEMPIO DI ESPRESSIONE REGOLARE

$L = \{w \mid w = a^{2k+1}$ (con k ≥ 0)$\}$, questo linguaggi è denotato da più espressioni regolari tra loro equivalenti:

- $(aa)^*a$
- $a(aa)^*$
- $(aa)^*a(aa)^*$
- etc

ESEMPIO DI ESPRESSIONE REGOLARE

prendiamo l'espressione regolare GOOO*GLE

cosa rappresenta Gooo*gle? Essendo un'espressione regolare rappresenta un linguaggio, e dato che Gooo*gle contiene o* allora l'espressione regolare rappresenta un linguaggi infinito, in quanto le parole del linguaggio possono essere ottenute prendendo o* nelle sue potenze, ossia, {ε, o, oo, ooo, oooo, …}, abbiamo quindi che Gooo*gle denota il seguente linguaggio:

$L = \{Google, Gooogle, Goooogle, Gooooogle, Goooooogle\} = \{Go^2gle, Go^3gle, Go^4gle, Go^5gle, Go^6gle, \}$

ESEMPIO DI ESPRESSIONE REGOLARE

Prendiamo il linguaggio di programmazione C e vediamo l'espressione regolare che definisce gli indentificatori di variabili, per il nostro esercizio l'identificatore di variabile (nome della variabile) è una stringa che può iniziale o con una lettera maiuscola o con una lettera minuscola, a patto che non sia ne un numero ne un carattere speciale, la prima lettera è definito da {A + B + C + … + Z + a + b + … + u + v + z}

Dal secondo carattere della stringa, in poi, viene accettato qualsiasi carattere (maiuscolo o minuscolo) e vengono accettati anche i numeri, la lunghezza dell'identificatore di variabile è lungo a piacere quindi ottengo che per costruire un identificatore di variabile devo servirmi della seguente espressione regolare:

{A + B + … + Z + a + b + … + z} · { A + B + … + Z + a + b + … + z + 0 + 1 + … + 9}*

lo star nella seconda parte della espressione algebrica mi consente di identificare anche parole con lunghezza 1, appartiene infatti al linguaggio degli identificatori di variabile sia la parola w = A, sia la parola w = z12A.

ESEMPIO DI ESPRESSIONE REGOLARE

prendiamo il linguaggio che permette la scrittura delle cifre in euro sugli assegni e analizziamo l'espressione regolare che definisce detto linguaggio:

la prima cifra di un assegno non può essere zero, quindi è definita da $\{1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9\}$

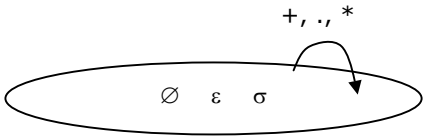
dalla seconda cifra in poi posso inserire un numero arbitrario di cifre a mio piacimento (sempre che abbia davvero un numero arbitrario di soldi con cui coprire l'assegno) inclusa la cifra "0", ossia $\{0 + 1 + \dots + 9\}^*$

devo a questo punto inserire il separatore degli euro dai cents, ossia devo inserire la virgola, infine devo permettere l'inserimento dei centesimi che devono comparire sugli assegni come due numeri, non tre e non uno. Ottengo quindi che per costruire l'importo di un assegno devo servirmi della seguente espressione regolare: $L = \{1 + \dots + 9\} \cdot \{0 + 1 + \dots + 9\}^* , \{0 + 1 + \dots + 9\}^2$

LE ESPRESSIONI REGOLARI SODDISFANO UNA PROPRIETÀ P

Per dimostrare che un' espressione regolare (ER) soddisfa una certa proprietà P si necessita del principio di induzione che si compone come segue:

- si necessita di dimostrare che P vale per le espressioni regolari base, ossia che P vale per $\varnothing$, ε, $\sigma \in \Sigma$.
- Supponendo per ipotesi che P vale per le espressioni regolari p e q allora si necessita di dimostrare che P vale anche per:
  - $p + q$
  - $p \cdot q$
  - p^*



devo quindi dimostrare che $\varnothing$, ε, σ soddisfano P e +, ·, * preservano la proprietà P.

TEOREMA DI KLEENE

RICORDANDO PRIMA CHE: esiste il TEOREMA di equivalenza tra grammatiche di tipo 3 e automi DFA:

1) se L è generato da G di tipo 3 $\Rightarrow \exists$ DFA che riconosce L
2) se L è riconosciuto da DFA $\Rightarrow \exists$ G di tipo 3 che genera L

TEOREMA

L è denotato da un'espressione regolare $\leftrightarrow$ L riconosciuto da DFA

1) se un linguaggio è denotato da un'espressione regolare $\Rightarrow$ esiste un automa DFA che lo definisce
2) se un automa DFA riconosce un linguaggio $\Rightarrow$ questo è denotato da un'espressione regolare

Il teorema quindi fornisce un'equivalenza tra le espressioni regolari e gli automi a stati finiti. Per asserire che quest'equivalenza esiste dobbiamo dimostrare i due punti del teorema di kleene.

DIMOSTRAZIONE:

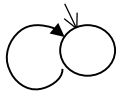
1) L DENOTATO DA ER $\Rightarrow \exists$ DFA CHE RICONOSCE L

Si utilizza la tecnica dimostrativa data per le espressioni regolari, in questo caso P (la proprietà) è l'esistenza di un automa DFA in grado di riconoscere il linguaggio L denotato dall'espressione regolare in oggetto,

P = DFA per linguaggio L

Sarà sufficiente provare che le espressioni regolari sono riconosciute da automa a stati finiti e che se A e B sono espressioni regolari riconosciute da automi a stati finiti allora anche $A \cup B$, $A \cdot B$ e A^* sono riconosciute da automi a stati finiti, per effettuare detta dimostrazione sarà necessario procedere per passi, dimostrando un'area alla volta:

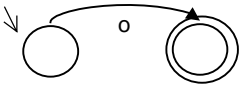
- per le espressioni regolari base:
  - dobbiamo esibire un automa per il linguaggio denotato dall'espressione regolare $\varnothing$, ossia $L = \varnothing$, l'automa sarà definito da un unico stato non finale, quindi da uno stato trappola


ER = $\varnothing$,

  - dobbiamo esibire un automa per il linguaggio denotato dall'espressione regolare ε, ossia $L = \{\varepsilon\}$, l'automa sarà definito da un unico stato finale, dal quale non partono archi, la transizione dettata dalla lettura di un qualsiasi simbolo porterà infatti nello stato trappola


ER = ε,

- dobbiamo esibire un automa per il linguaggio denotato dall'espressione regolare o $\in \Sigma$, ossia L = {o}, l'automa sarà definito da due stati, il primo sarà lo stato iniziale e il secondo sarà lo stato finale, la transizione dal primo al secondo stato sarà definita dall'inserimento del carattere o, la transizione dettata dalla lettura di un qualsiasi simbolo $\alpha \neq \sigma$ porterà invece nello stato trappola

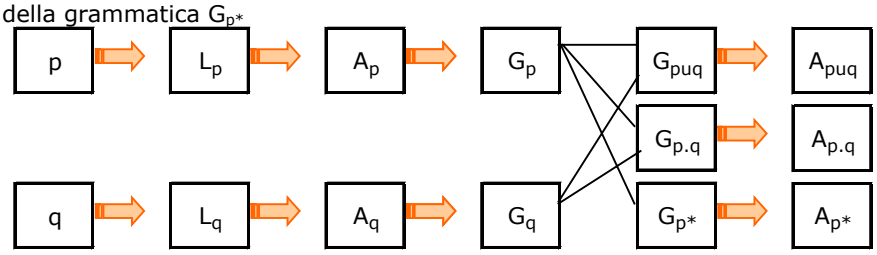


ER = o $\in \Sigma$,

- supponiamo per ipotesi che esistano due automi DFA (A_p e A_q) dove A_p è l'automa DFA in grado di riconoscere L_p (linguaggio denotato dall'ER p) e A_q è l'automa DFA in grado di riconoscere L_q (linguaggio denotato dall'ER q), dal teorema di equivalenza tra G e DFA sappiamo che: se L è riconosciuto da DFA $\Rightarrow \exists$ G di tipo 3 che genera L, dato che per ipotesi esistono due automi DFA (A_p e A_q) rispettivamente per L_p e $L_q \Rightarrow \exists$ G_p di tipo 3 che genera L_p e G_q di tipo 3 che genera L_q.

ER		L denotato da ER		DFA per L		G equivalente a DFA
p	⇨	L_p	⇨	A_p	⇨	G_p
q	⇨	L_q	⇨	A_q	⇨	G_q

  - si necessita di dimostrare l'esistenza degli automi $A_{p\cup q}$ in grado di riconoscere il linguaggio $L_{p\cup q}$ denotato dall'espressione regolare p + q, dal teorema di equivalenza tra G e DFA sappiamo che: se L è riconosciuto da DFA $\Rightarrow \exists$ G di tipo 3 che genera L, quindi dimostrare l'esistenza dell'automa $A_{p\cup q}$ equivale a dimostrare l'esistenza della grammatica $G_{p\cup q}$
  - si necessita di dimostrare l'esistenza degli automi $A_{p\cdot q}$ in grado di riconoscere il linguaggio $L_{p\cdot q}$ denotato dall'espressione regolare p · q, dal teorema di equivalenza tra G e DFA sappiamo che: se L è riconosciuto da DFA $\Rightarrow \exists$ G di tipo 3 che genera L, quindi dimostrare l'esistenza dell'automa $A_{p\cdot q}$ equivale a dimostrare l'esistenza della grammatica $G_{p\cdot q}$
  - si necessita di dimostrare l'esistenza degli automi A_{p^*} in grado di riconoscere il linguaggio L_{p^*} denotato dall'espressione regolare p*, dal teorema di equivalenza tra G e DFA sappiamo che: se L è riconosciuto da DFA $\Rightarrow \exists$ G di tipo 3 che genera L, quindi dimostrare l'esistenza dell'automa A_{p^*} equivale a dimostrare l'esistenza della grammatica G_{p^*}



Nello schema sopra disegnato tutti i passaggi denotati da una freccia arancione sono passaggi dimostrati da teoremi, l'unico passaggio non dimostrato è la costruzione delle grammatiche $G_{p\cup q}$, $G_{p\cdot q}$ e G_{p^*}, l'esistenza di queste tre grammatiche e la loro costruzione vanno effettuate in maniera esplicita.

RICORDANDO PRIMA CHE: Le grammatiche di tipo 3 sono identificate da regole di produzione che assumono una delle seguenti quattro tipologie di forme tra loro equivalenti:

1) $A \to x$ oppure $A \to yB$ (dette lineari destre)
2) $A \to \sigma$ oppure $A \to \sigma B$ oppure $A \to \varepsilon$
3) $A \to \sigma B$ oppure $A \to \varepsilon$
4) $A \to \sigma B$ oppure $A \to \sigma$

dove: A, B $\in$ M, x, y $\in \Sigma^*$, $\sigma \in \Sigma$

RICORDANDO INOLTRE CHE: le quattro tipologie di forme si dicono equivalenti in quanto possono essere trasformate da una forma all'altra senza modificare il linguaggio che la grammatica genera

Sia $G_p = <\Sigma_p, M_p, P_p, S_p>$ con le regole di produzione di P_p nella forma $A \to \varepsilon \mid A \to \sigma B$,

Quali sono le parole generate da G_p? $L(G_p) = \{w \in \Sigma^* \mid S_p \Rightarrow^* w\}$

Sia $G_q = <\Sigma_q, M_q, P_q, S_q>$ con le regole di produzione di P_q nella forma $A \to \varepsilon \mid A \to \sigma B$,

Dove $\Sigma_p = \Sigma_q$ che d'ora in poi indicheremo con Σ

Quali sono le parole generate da G_q? $L(G_q) = \{v \in \Sigma^* \mid S_q \Rightarrow^* v\}$

Costruiamo $G_{p\cup q}$:

- $G_{p\cup q}$ è la grammatica che deve generare il linguaggio $L_{p\cup q} = L_p \cup L_q$, ossia è la grammatica in grado di generare sia le parole generate dalla grammatica G_p sia le parole generate dalla grammatica G_q.
  - Quali parole dovrà generare $G_{p\cup q}$? $L(G_{p\cup q}) = \{z \in \Sigma^* \mid z \in L_p \text{ oppure } z \in L_q\}$
  - Di che cosa ho bisogno per generare parole del linguaggio L_p (w)?
    - Ho bisogno delle regole di produzione di P_p e dell'assioma S_p, infatti, se prendo S_p e applico le regole di P_p ottengo tutte e sole quelle parole $w \in L_p$.
  - Di che cosa ho bisogno per generare parole del linguaggio L_q (v)?
    - Ho bisogno delle regole di produzione di P_q e dell'assioma S_q, infatti, se prendo S_q e applico le regole di P_q ottengo tutte e sole quelle parole $v \in L_q$.
  - Come faccio a creare regole di produzione che mi permettano di generare o $w \in L_p$ o $v \in L_q$?
    - devo porre un nuovo assioma S che mi consentirà di poter andare o in S_p o in S_q a seconda che stia creando una parola di L_p o una parola di L_q. Devo quindi inserire nell insieme $P_{p\cup q} = P_p \cup P_q$ anche le regole $S \to S_p$ e $S \to S_q$. domanda: le regole di produzioni appena inserite sono di tipo 3? Si in quanto sono regole di produzione lineari a destra della forma $A \to yB$ con $y = \varepsilon$.
  - Come sarà quindi l'insieme dei metasimboli $M_{p\cup q}$?
    - $M_{p\cup q} = M_p \cup M_q \cup \{S\}$, questo è possibile $\leftrightarrow M_p \cap M_q \cap \{S\} = \varnothing$, ossia sarà possibile effettuare l'unione dei due insiemi solo se questi risulteranno disgiunti. Perché devono essere disgiunti? Se i due insiemi non fossero disgiunti, applicando le regole di produzione di $P_{p\cup q}$ potrei effettuare transizioni su metasimboli di M_p con regole di produzione di P_q o viceversa potrei effettuare transizioni su metasimboli di M_q con regole di prodizione di P_p. Se $M_p \cap M_q \cap \{S\} \neq \varnothing$ allora devo rinominare sia in M_q che in P_q tutti quei metasimboli che appartengono all'insieme intersezione.

  $G_{p\cup q} = <\Sigma_{p\cup q}, M_{p\cup q}, P_{p\cup q}, S_{p\cup q}>$, dove

  $\Sigma_{p\cup q} = \Sigma_p = \Sigma_q = \Sigma$

  $M_{p\cup q} = M_p \cup M_q \cup \{S_{p\cup q}\}$ # a patto che $M_p \cap M_q \cap \{S_{p\cup q}\} = \varnothing$

  $P_{p\cup q} = P_p \cup P_q \cup \{S_{p\cup q} \to S_p, S_{p\cup q} \to S_q\}$

  $S_{p\cup q}$ è il nuovo assioma

ESERCIZIO

Sia $G_p = <\Sigma_p = \{a, b\}, M_p = \{S'\}, P_p = \{S' \to \varepsilon \mid aS'\}, S'>$, quali parole genera G_p? $L(G_p) = a^*$

Sia $G_q = <\Sigma_q = \{a, b\}, M_q = \{S''\}, P_q = \{S'' \to \varepsilon \mid bS''\}, S''>$, quali parole genera G_p? $L(G_p) = b^*$

G_p e G_q sono di tipo 3? Si, le regole di produzione di P_p e P_q risultano essere tutte nella forma $A \to \varepsilon \mid A \to \sigma B$

Costruzione di $G_{p\cup q}$, grammatica che deve generare il linguaggio $L_{p\cup q} = L_p \cup L_q = a^* \cup b^* = a^* + b^*$

$G_{p\cup q} = <\Sigma_{p\cup q}, M_{p\cup q}, P_{p\cup q}, S_{p\cup q}>$, dove

$\Sigma_{p\cup q} = \Sigma_p = \Sigma_q = \{a, b\}$

$M_{p\cup q} = M_p \cup M_q \cup \{S_{p\cup q}\} = \{S'\} \cup \{S''\} \cup \{S\} = \{S', S'', S\}$ # dato che $\{S'\} \cap \{S''\} \cap \{S\} = \varnothing$

$P_{p\cup q} = P_p \cup P_q \cup \{S_{p\cup q} \to S_p | S_q\} = \{S' \to \varepsilon | aS'\} \cup \{S'' \to \varepsilon | bS''\} \cup \{S \to S' | S''\} =$

$= \{$ 1) $S' \to \varepsilon$, 2) $S' \to aS'$, 3) $S'' \to \varepsilon$, 4) $S'' \to bS''$, 5) $S \to S'$, 6) $S \to S''\}$

$S_{p\cup q} = S$

$G_{p\cup q} = <\Sigma_{p\cup q} = \{a, b\}, M_{p\cup q} = \{S', S'', S\}, P_{p\cup q} = \{S' \to \varepsilon \mid aS', S'' \to \varepsilon \mid bS'', S \to S' \mid S''\}, S>$

$G_{p\cup q}$ è di tipo 3? Si, le regole di prod. sono lineari a destra, infatti $S \to S' | S''$ sono nella forma $A \to yB$ con $y = \varepsilon$.

Test di $G_{p\cup q}$: generare $a^3 \in L_p$, generare $a^2b \notin L_p$, $a^2b \notin L_q$, generare $b^2 \in L_q$

a^3: $S \Rightarrow^5 S' \Rightarrow^2 aS' \Rightarrow^2 aaS' \Rightarrow^2 aaaS' \Rightarrow^1 aaa$ $\quad aaa \in L_{p\cup q}$

a^2b: $S \Rightarrow^5 S' \Rightarrow^2 aS' \Rightarrow^2 aaS'$ $\quad aab \notin L_{p\cup q}$ infatti non esiste la regola di produzione $S' \to S''$

b^2: $S \Rightarrow^6 S'' \Rightarrow^4 bS'' \Rightarrow^4 bbS'' \Rightarrow^3 bb$ $\quad bb \in L_{p\cup q}$

data ora $G_{p\cup q}$ generiamo l'automa $A^{p\cup q}$:

$A^{p\cup q} = <\Sigma^A, Q^A, q_0{}^A, \delta^A, F^A>$, dove

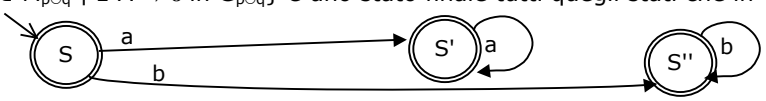
$\Sigma^A = \Sigma_{p\cup q}$ l'alfabeto dei simboli terminali non cambia rispetto a quello della grammatica

$Q^A = M_{p\cup q}$ gli stati dell'automa sono i metasimboli della grammatica $G_{p\cup q}$

$q_0{}^A = S_{p\cup q}$ lo stato iniziale dell'automa è determinato dall'assioma di partenza per la grammatica

δ^A: $\exists\, q \to \sigma p \in P_{p\cup q} \Rightarrow \delta(q, \sigma) = p$ per ogni produzione del tipo $q \to \sigma p$ definisco la regola $\delta(q, \sigma) = p$ in A

$F^A = \{A \in M_{p\cup q} \mid \exists\, A \to \varepsilon \text{ in } G_{p\cup q}\}$ è uno stato finale tutti quegli stati che in $G_{p\cup q}$ hanno la regola $A \to \varepsilon$



Test di $A^{p\cup q}$: generare $a^3 \in L_p$, generare $a^2b \notin L_p$, $a^2b \notin L_q$, generare $b^2 \in L_q$

"a^3" è accettata dall'automa? si in quanto esiste un cammino accettante, $S \xrightarrow{a} S' \xrightarrow{a} S' \xrightarrow{a} S'$

"a^2b" è accettata dall'automa? NO, non esistono cammini accettanti, $S \xrightarrow{a} S' \xrightarrow{a} S' \xrightarrow{b} X$

"b^2" è accettata dall'automa? si in quanto esiste un cammino accettante, $S \xrightarrow{b} S'' \xrightarrow{b} S''$

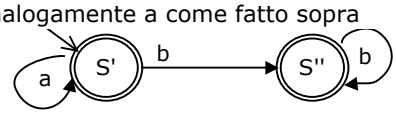
Costruiamo $G_{p\cdot q}$:

- $G_{p\cdot q}$ è la grammatica che deve generare il linguaggio $L_{p\cup q} = L_p \cdot L_q$, ossia è la grammatica in grado di generare parole generate dalla grammatica G_p a cui vengono concatenate parole generate da G_q.
  - Quali parole dovrà generare $G_{p\cdot q}$? $L(G_{p\cdot q}) = \{z \in \Sigma^* \mid z = x\cdot y \text{ t.c. } x \in L_p \text{ e } y \in L_q\}$
  - Di che cosa ho bisogno per generare parole del linguaggio L_p (x)?
    - Delle regole di produzione di P_p e dell'assioma S_p per costruire tutte e sole parole $x \in L_p$.
  - Di che cosa ho bisogno per generare parole del linguaggio L_q (y)?
    - Delle regole di produzione di P_q e dell'assioma S_q per costruire tutte e sole parole $y \in L_q$.
  - Come faccio a creare regole di produzione che mi permettano di generare $z = x \cdot y$?
    - devo costruire parole di L_p e all'ultima produzione concatenare parole di L_q, come posso effettuare questo passaggio? Sappiamo che le parole $x \in L_p$ sono prodotte tramite la seguente transizione:
      $S_p \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow xX \Rightarrow x$
      mentre le parole $y \in L_q$ sono prodotte tramite la seguente transizione:
      $S_q \Rightarrow y_1 \Rightarrow y_2 \Rightarrow \dots \Rightarrow yY \Rightarrow y$
      qual è l'ultima regola di produzione che ho applicato per passare da xX a $x \in L_p$? Sicuramente esiste una regola di produzione $X \to \varepsilon \in P_p$, dato che non voglio fermarmi bensì voglio iniziare la creazione della parola $y \in L_q$ come posso effettuare questo passaggio? $X \to \varepsilon$ deve essere sostituita dalla regola di produzione $X \to S_q$ questa regola mi consente di effettuare il seguente passaggio:
      $S_p \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow xX \Rightarrow xS_q \Rightarrow xy_1 \Rightarrow xy_2 \Rightarrow \dots \Rightarrow xyY \Rightarrow xy$
      Dove xy = z ossia la parola che volevamo produrre
  - Come sarà quindi l'insieme delle regole di produzione $P_{p\cdot q}$?
    - $P_{p\cdot q} = P_p \setminus \{A\to\varepsilon\} \cup P_q \cup \{A \to S_q \ \forall\ A \to \varepsilon \in P_p\}$, ossia $P_{p\cdot q}$ è l'insieme delle regole di produzione P_p cui vengono tolte le regole di produzione $A \to \varepsilon$, unito le regole di produzione P_q cui vengono aggiunte regole nella forma $A \to S_q$ $\forall$ regola $A \to \varepsilon$ precedentemente eliminata dall'insieme P_p.
  - Come sarà quindi l'insieme dei metasimboli $M_{p\cdot q}$?
    - $M_{p\cdot q} = M_p \cup M_q$, questo è possibile $\leftrightarrow M_p \cap M_q = \varnothing$, ossia sarà possibile effettuare l'unione dei due insiemi se e solo se questi risulteranno disgiunti. In caso contrario si necessiterà di rinominare sia in M_q che in P_q tutti quei metasimboli che appartengono all'insieme intersezione.

$G_{p\cdot q} = <\Sigma_{p\cdot q}, M_{p\cdot q}, P_{p\cdot q}, S_{p\cdot q}>$, dove
$\Sigma_{p\cdot q} = \Sigma_p = \Sigma_q = \Sigma$
$M_{p\cdot q} = M_p \cup M_q$ # a patto che $M_p \cap M_q = \varnothing$
$P_{p\cdot q} = P_p \setminus \{A \to \varepsilon\} \cup P_q \cup \{A \to S_q \mid \forall\ A \to \varepsilon \in P_p\}$
$S_{p\cup q} = S_p$

ESERCIZIO

Sia $G_p = <\Sigma_p = \{a, b\}, M_p = \{S'\}, P_p = \{S' \to \varepsilon \mid aS'\}, S'>$, quali parole genera G_p? $L(G_p) = a^*$
Sia $G_q = <\Sigma_q = \{a, b\}, M_q = \{S''\}, P_q = \{S'' \to \varepsilon \mid bS''\}, S''>$, quali parole genera G_p? $L(G_p) = b^*$
Costruzione di $G_{p\cdot q}$, grammatica che deve generare il linguaggio $L_{p\cdot q} = L_p \cdot L_q = a^* \cdot b^* = a^*b^*$
$G_{p\cdot q} = <\Sigma_{p\cdot q}, M_{p\cdot q}, P_{p\cdot q}, S_{p\cdot q}>$, dove
$\Sigma_{p\cdot q} = \Sigma_p = \Sigma_q = \{a, b\}$
$M_{p\cdot q} = M_p \cup M_q = \{S'\} \cup \{S''\} = \{S', S''\}$ # dato che $\{S'\} \cap \{S''\} = \varnothing$
$P_{p\cdot q} = P_p \setminus \{A \to \varepsilon\} \cup P_q \cup \{A \to S_q \mid \forall\ A \to \varepsilon\} = \{S' \to \varepsilon, S' \to aS'\} \setminus \{S' \to \varepsilon\} \cup \{S''\to\varepsilon|bS''\} \cup \{S' \to S''\}$
$= \{S' \to aS'\} \cup \{S' \to S'', S'' \to \varepsilon \mid bS''\} = \{1)\ S' \to S'',\ 2)\ S' \to aS',\ 3)\ S'' \to \varepsilon,\ 4)\ S'' \to bS''\}$
$S_{p\cdot q} = S'$
$G_{p\cdot q} = <\Sigma_{p\cup q} = \{a, b\}, M_{p\cup q} = \{S', S''\}, P_{p\cup q} = \{S' \to S'' \mid aS', S'' \to \varepsilon \mid bS''\}, S'>$
Test di $G_{p\cdot q}$: generare a^3, generare a^2b, generare b^2, generare ab^2a
a^3: $S' \Rightarrow^2 aS' \Rightarrow^2 aaS' \Rightarrow^2 aaaS' \Rightarrow^1 aaaS'' \Rightarrow^3 aaa$ $aaa \in L_{p\cdot q}$
a^2b: $S' \Rightarrow^2 aS' \Rightarrow^2 aaS' \Rightarrow^1 aaS'' \Rightarrow^4 aabS'' \Rightarrow^3 aab$ $aab \in L_{p\cdot q}$
b^2: $S' \Rightarrow^1 S'' \Rightarrow^4 bS'' \Rightarrow^4 bbS'' \Rightarrow^3 bb$ $bb \in L_{p\cdot q}$
ab^2a: $S' \Rightarrow^2 aS' \Rightarrow^1 aS'' \Rightarrow^4 abS'' \Rightarrow^4 abbS''$ $abba \notin L_{p\cdot q}$
data ora $G_{p\cdot q}$ generiamo l'automa $A^{p\cdot q}$:
$A^{p\cdot q} = <\Sigma^A, Q^A, q_0{}^A, \delta^A, F^A>$, definito analogamente a come fatto sopra



Test di $A^{p\cdot q}$: generare a^3, generare a^2b, generare b^2, generare ab^2a
"a^3" è accettata dall'automa? si in quanto esiste un cammino accettante, $S' \xrightarrow{a} S' \xrightarrow{a} S' \xrightarrow{a} S'$
"a^2b" è accettata dall'automa? si in quanto esiste un cammino accettante, $S' \xrightarrow{a} S' \xrightarrow{a} S' \xrightarrow{b} S''$
"b^2" è accettata dall'automa? si in quanto esiste un cammino accettante, $S' \xrightarrow{b} S'' \xrightarrow{b} S''$
"ab^2a" è accettata dall'automa? in quanto l'automa si blocca, $S' \xrightarrow{a} S' \xrightarrow{b} S'' \xrightarrow{b} S'' \xrightarrow{a} X$

Costruiamo G_{p^*}:

- G_{p^*} è la grammatica che deve generare il linguaggio $L_{p^*} = (L_p)^*$, ossia è la grammatica in grado di generare parole generate dalla grammatica G_p prese un numero infinito di volte in qualsiasi combinazione.

  Quali parole dovrà generare G_{p^*}? $L(G_{p^*}) = L_p^* = L_p^0 \cup L_p^1 \cup \dots \cup L_p^\infty = \{\varepsilon\} \cup L_p^1 \cup \dots \cup L_p^\infty = \{\varepsilon\} \cup L^+$

  Di che cosa ho bisogno per generare parole del linguaggio L_p^+?

  Ho bisogno delle regole di produzione di P_p e dell'assioma S_p, infatti, se prendo S_p e applico le regole di P_p ottengo tutte e sole quelle parole $x \in L_p$.

  A questo punto ho bisogno di tutte le parole $x \in L_p$ da concatenare tra loro in tutti i modi possibili, l'idea è quindi:

  $v \in L_p^+ \leftrightarrow v = w_1 \cdot w_2 \cdot w_3 \cdot \dots \cdot w_\infty$ con $w_i \in L_p$

  ossia:

  $S_p \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow wA \Rightarrow w$

  Ora invece di fermarmi debvo generare di nuovo un'altra parola w quindi inserisco una regola di produzione della forma $A \rightarrow S_p$ per ogni regola di produzione della forma $A \rightarrow \varepsilon$ esistente nella grammatica.

  La regola $A \rightarrow \varepsilon$ deve essere cancellata? NO

  Perché non devo cancellare le regole di produzione nella forma $A \rightarrow \varepsilon$? Perché la cancellazione di dette regole di produzione non mi consentirebbero di terminare la generazione delle parole di L_p^+.

  A questo punto si necessita di un ultimo controllo: se L_p^+ contiene la parola vuota (ε) allora abbiamo terminato altrimenti dobbiamo introdurre la parola vuota e come facciamo? Come si introduce la parola vuota? Si aggiunge la regola di produzione $S \rightarrow \varepsilon$

$G_{p^*} = <\Sigma_{p^*}, M_{p^*}, P_{p^*}, S_{p^*}>$, dove

$\Sigma_{p^*} = \Sigma_p = \Sigma$

$M_{p^*} = M_p$

$P_p^+ = P_p \cup \{\forall\ A \rightarrow \varepsilon \in P_p,\ A \rightarrow S_p \mid A \in M_p\}$ (se L_p^+ non contiene ε, $P_{p^*} = P_p^+ \cup \{S_p \rightarrow \varepsilon\}$)

$S_{p^*} = S_p$

ESERCIZIO

Sia $G_p = <\Sigma_p = \{a\}, M_p = \{S, A, B\}, P_p = \{S \rightarrow aA, A \rightarrow aB, B \rightarrow \varepsilon\}, S>$, quali parole genera G_p? $L(G_p) = a^2$

Costruzione di G_{p^*}, grammatica che deve generare il linguaggio $L_{p^*} = (L_p)^* = (a^2)^*$

Per poter effettuare la costruzione di G_p^*, dobbiamo prima effettuare la costruzione di G_p^+ grammatica che deve generare il linguaggio $L_p^+ = (L_p)^+ = \{(a^2)^i \mid i \geq 1\}$

$G_{p^*} = <\Sigma_{p^*}, M_{p^*}, P_{p^*}, S_{p^*}>$, dove

$\Sigma_{p^*} = \Sigma_p = \{a\}$

$M_{p^*} = M_p = \{S, A, B\}$

$P_p^+ = P_p \cup \{\forall\ A \rightarrow \varepsilon \in P_p,\ A \rightarrow S_p\} = \{S \rightarrow aA, A \rightarrow aB, B \rightarrow \varepsilon\} \cup \{B \rightarrow S\} = \{S \rightarrow aA, A \rightarrow aB, B \rightarrow \varepsilon \mid S\}$

dato che L_p^+ non contiene la parola vuota $\varepsilon \Rightarrow P_{p^*} = P_p^+ \cup \{S \rightarrow \varepsilon\} =$

$P_p^* = \{S \rightarrow aA, A \rightarrow aB, B \rightarrow \varepsilon \mid S\} \cup \{S \rightarrow \varepsilon\} = \{1)\ S \rightarrow \varepsilon,\ 2)\ S \rightarrow aA,\ 3)\ A \rightarrow aB,\ 4)\ B \rightarrow S,\ 5)\ B \rightarrow \varepsilon\}$

$S_{p^*} = S_p = S$

$\Sigma_{p \cdot q} = \Sigma_p = \Sigma_q = \{a, b\}$

$M_{p \cdot q} = M_p \cup M_q = \{S'\} \cup \{S''\} = \{S', S''\}$ # dato che $\{S'\} \cap \{S''\} = \varnothing$

$G_{p^*} = <\Sigma_{p^*} = \{a\}, M_{p^*} = \{S, A, B\}, P_{p^*} = \{S \rightarrow aA, A \rightarrow aB, B \rightarrow \varepsilon \mid S\}, S>$

Test di G_{p^*}: generare a^4, generare a^3, generare a^2, generare ε

a^4: $S \Rightarrow^2 aA \Rightarrow^3 aaB \Rightarrow^4 aaS \Rightarrow^2 aaaA \Rightarrow^3 aaaaB \Rightarrow^5 aaaa$ $\quad aaaa \in L_{p^*}$

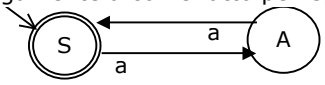
a^3: $S \Rightarrow^2 aA \Rightarrow^3 aaB \Rightarrow^4 aaS \Rightarrow^2 aaaA$ $\quad aaa \notin L_{p^*}$

a^2: $S \Rightarrow^2 aA \Rightarrow^3 aaB \Rightarrow^5 aa$ $\quad aa \in L_{p^*}$

ε : $S \Rightarrow^1 \varepsilon$ $\quad \varepsilon \in L_{p^*}$

data ora G_{p^*} generiamo l'automa A^{p^*}:

$A^{p^*} = <\Sigma^A, Q^A, q_0^A, \delta^A, F^A>$, definito analogamente a come fatto per $G_{p \cup q}$



Test di A^{p^*}: generare a^4, generare a^3, generare a^2, generare ε

"a^3" è accettata dall'automa? si in quanto esiste un cammino accettante, $S \xrightarrow{a} A \xrightarrow{a} S \xrightarrow{a} A \xrightarrow{a} S$

"a^3" è accettata dall'automa? NO in quanto NON esiste un cammino accettante, $S \xrightarrow{a} A \xrightarrow{a} S \xrightarrow{a} A$

"a^2" è accettata dall'automa? si in quanto esiste un cammino accettante, $S \xrightarrow{a} A \xrightarrow{a} S$

"ε" è accettata dall'automa? si in quanto esiste un cammino accettante, S

2) L RICONOSCIUTO DA DFA ⇒ ∃ ER CHE DENOTA L

dobbiamo ricavare un espressione regolare ER partendo da un automa a stati finiti. Questa dimostrazione si basa sulla definizione di sistema di equazioni, dove le incognite sono i linguaggi, che chiameremo X_i, e dove la risoluzione del sistema ci consente di trovare il linguaggio riconosciuto dall'automa specificato sotto forma di espressione regolare. Siamo interessati ad esprimere X_0, il quale rappresenta proprio il linguaggio riconosciuto dall'automa fornito in input, detto linguaggio dovrà risultare essere espresso in forma di ER.

COSA SONO I LINGUAGGI X_i?

$A = <\Sigma, Q = \{q_0, q_1, q_2, ..., q_k\}, q_0, \delta, F>$, automa fornito in input
Come si ricava l'espressione regolare che denota il linguaggio riconosciuto da A?
A è un automa che implicitamente definisce altri automi, come? È sufficiente modificare lo stato iniziale, da q_0 in un qualsiasi altro stato $q_i \in Q$.

DA A AGLI A_i

Ad A è possibile associare k automi da A_1 ad A_k, dove k indica il numero di stati di Q, associabili come segue:

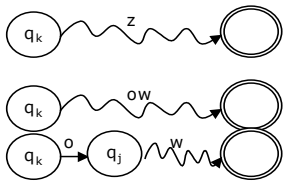
$A \rightarrow$ associo $\rightarrow A_1$ A_1 è simile ad A ad eccezione dello stato iniziale che per A_1 è q_1, $A_1 = <\Sigma, Q, q_1, \delta, F>$
$A \rightarrow$ associo $\rightarrow A_i$ A_i è simile ad A ad eccezione dello stato iniziale che per A_i è q_i, $A_i = <\Sigma, Q, q_i, \delta, F>$
$A \rightarrow$ associo $\rightarrow A_k$ A_k è simile ad A ad eccezione dello stato iniziale che per A_k è q_k, $A_k = <\Sigma, Q, q_k, \delta, F>$

DAGLI A_i AGLI X_i

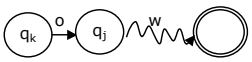
A ciascun automa A_i appena definito posso associare il linguaggio che l'automa A_i riconosce, detto linguaggio viene indicato con la dicitura X_i e l'associazione viene effettuata come segue:
X_0 è il linguaggio riconosciuto da A, dove A è l'automa avente stato iniziale in q_0
X_1 è il linguaggio riconosciuto da A_1, dove A_1 è l'automa avente stato iniziale in q_1
X_i è il linguaggio riconosciuto da A_i, dove A_i è l'automa avente stato iniziale in q_i
X_k è il linguaggio riconosciuto da A_k, dove A_k è l'automa avente stato iniziale in q_k

ESPRIMERE X_i IN FUNZIONE DEGLI ALTRI X_h

Si necessita ora di esprimere X_i in funzione degli altri linguaggi al fine di ottenere delle equazioni nella forma
$X_i = \sigma X_0 + \sigma X_1 + \ldots + \sigma X_k$ dove $\sigma \in \Sigma$
Queste saranno le equazioni che apparterranno al sistema da risolvere, a noi interesserà poter calcolare X_0 ottenendo l'espressione regolare che lo definisce.
Regole per esprimere un linguaggio in funzione degli altri linguaggi:
1) $z \in X_k$ ($z \neq \varepsilon$, $z = \sigma w$) $\leftrightarrow \delta(q_k, z) \in F$, ossia $\delta(q_j, w) \in F$, ossia $w \in X_j$
2) $\varepsilon \in X_k \leftrightarrow q_k \in F$



A) Una parola z (non vuota) appartiene a $X_k \leftrightarrow$ partendo dallo stato q_k e leggendo z si arriva in uno stato finale, $\delta(q_k, z) \in F$
B) Se $z \neq \varepsilon \Rightarrow$ z è esprimibile come il prodotto di concatenazione tra σ e w con ($\sigma \in \Sigma$, $w \in \Sigma^*$), ossia $z = \sigma w$
C) cosa significa dire che $\delta(q_k, \sigma w) \in F$? significa dire che $\delta(\delta(q_k, \sigma), w) \in F$, dove sappiamo che $\delta(q_k, \sigma)$ è uno stato dell'automa e lo possiamo indicare con q_j

$\delta(q_k, \sigma w) \in F \leftrightarrow \delta(q_j, w) \in F$, 

ossia $\leftrightarrow$ 

REGOLE PER LA COSTRUZIONE DELLE EQUAZIONI

regole per la costruzione dell'equazione una volta scritto il linguaggio in funzione di tutti gli altri linguaggi
X_k risulterà essere uguale all'unione dei linguaggi σX_j con $\sigma \in \Sigma$ e su tutti quei j tali per cui leggendo σ raggiungo uno stato q_j. Come faccio ad esprimere ciò sotto forma di formula matematica?

$X_k = \bigcup_{\sigma \in \Sigma,\ \{j \mid \delta(q_k, \sigma) = q_j\}} \sigma X_j$ (+ ε) se $q_k \in F$

EQUAZIONE TIPO PER LA SOLUZIONE

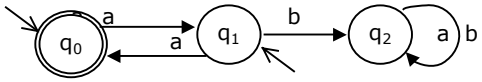
L'equazione tipo a cui far riferimento per la soluzione è: $X = AX + B$
dove X è l'incognita, A e B sono dei linguaggi specificati
La soluzione di $X = AX + B$ è una soluzione nota ed è A^*B
Dimostrazione che $X = AX + B = A^*B$

- sostituisco ad X il valore A^*B nell'espressione $X = AX + B \Rightarrow A^*B = A(A^*B) + B$
- chiusura di Kleene: $A^* = \cup$ Ai (con i che va da 0 a ∞) $\Rightarrow A^*B = A((\cup Ai)B) + B$
- proprietà associativa: $a \cdot (b \cdot c) = (a \cdot b) \cdot c \Rightarrow A^*B = (A(\cup Ai))B + B$ (con i che va da 0 a ∞)
- proprietà di assorbimento $L \cdot (\cup Li) = \cup L \cdot Li = \cup L\ i+1 \Rightarrow A^*B = (\cup Ai+1)B + B$ (con i che va da 0 a ∞)
- sostituisco ad i + 1 la variabile j per comodità $\Rightarrow A^*B = (\cup Aj)B + B$ (con j che va da 1 a ∞)
- chiusura di Kleene: $\cup$ Ai (con i che va da 1 a ∞) = A+ $\Rightarrow A^*B = (A+B) + B$
- raccolgo B ricordandomi che è il linguaggio suffisso quindi va raccolto in fondo $\Rightarrow A^*B = (A+ + \{\varepsilon\}) \cdot B$
- chiusura di Kleene: $A+ + \{\varepsilon\} = A^* \Rightarrow A^*B = (A^*) \cdot B$

$A^*B = A^*B$ vera quindi ho dimostrato che A^*B è proprio la soluzione dell'equazione

ESEMPIO

Sia A un automa a 3 stati definito come segue:

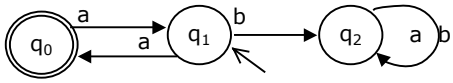
A = 

A è osservabile

X_0 è il linguaggio riconosciuto da A

$X_0 = \bigcup_{\sigma \in \Sigma,\ \{j \mid \delta(q_0, \sigma) = q_j\}} \sigma X_j + \varepsilon$ dato che $q_0 \in F$

Com'è definito l'automa A_1? Devo definire q_1 come stato iniziale, ossia

A_1 = 

A_1 è osservabile

X_1 è il linguaggio riconosciuto da A_1 che deve essere espresso in funzione di X_0 e X_2, ossia X_1 è il linguaggio X_0 dove, ad ogni parola di X_0 pongo quale prefisso il simbolo "a" unito al linguaggio X_2 dove, ad ogni parola di X_2 pongo quale prefisso il simbolo "b", ossia:

$X_1 = aX_0 + bX_2$

Da dove nasce questa equazione?

Partiamo con definire cos'è il linguaggi X_1

$X_1 = \{z \in \Sigma^+ \mid z = \sigma w \text{ dove } w \in \Sigma^*\}$

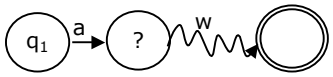
Per ora abbiamo solo affermato che X_1 è l'insieme di parole non vuote (infatti $z \in \Sigma^+$ e non a Σ^*) definite come concatenazione di un simbolo con una parola appartenente a Σ^*.

Ora devo inserire la condizione che, partendo dallo stato iniziale q_1 e leggendo la parola σw devo arrivare in uno stato finale, si ottiene quindi la seguente definizione di X_1:

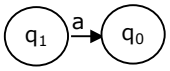
$X_1 = \{z \in \Sigma^+ \mid z = \sigma w,\ w \in \Sigma^* \text{ e } \delta(q_1, \sigma w) \in F\}$

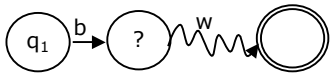
Che valori può assumere $\sigma \in \Sigma$? Dato che $\Sigma = \{a, b\}$ σ può valere o "a" o "b", possiamo quindi vedere X_1 non più quale l'insieme di parola appartenenti a Σ^+ con prefisso σ ma distinguere i casi in cui ho prefisso "a" e i casi in cui ho prefisso "b", X_1 risulterà quindi l'insieme di parole non vuote, tali per cui, partendo dallo stato iniziale q_1 e leggendo la parola aw devo arrivare in uno stato finale e analogamente partendo dallo stato iniziale q_1 e leggendo la parola bw devo arrivare in uno stato finale, è quindi possibile ridefinire X_1 come segue:

$X_1 = \{aw \in \Sigma^+ \mid \delta(q_1, aw) \in F\} \cup \{bw \in \Sigma^+ \mid \delta(q_1, bw) \in F\}$

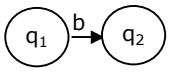
Cosa significa $\delta(q_1, aw) \in F$? 

dire $\delta(q_1, aw) \in F$ equivale a dire $\delta(q_0, w) \in F$

cos'è $\delta(q_1, aw)$? È lo stato q_0, ossia 

Cosa significa $\delta(q_1, bw) \in F$? 

dire $\delta(q_1, bw) \in F$ equivale a dire $\delta(q_2, w) \in F$

cos'è $\delta(q_1, bw)$? È lo stato q_2, ossia 

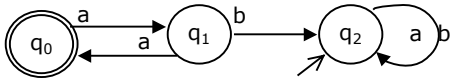
$X_1 = \{aw \in \Sigma^+ \mid \delta(q_1,aw) \in F\} \cup \{bw \in \Sigma^+ \mid \delta(q_1,bw) \in F\} = a\{w \in \Sigma^+ \mid \delta(q_0, w) \in F\} \cup b\{w \in \Sigma^+ \mid \delta(q_2, w) \in F\}$

Cos'è $\{w \in \Sigma^+ \mid \delta(q_0, w) \in F\}$? è l'insieme di parole riconosciute partendo da q_0, ossia X_0

Cos'è $\{w \in \Sigma^+ \mid \delta(q_2, w) \in F\}$? è l'insieme di parole riconosciute partendo da q_2, ossia X_2

$X_1 = a\{w \in \Sigma^+ \mid \delta(q_0, w) \in F\} \cup b\{w \in \Sigma^+ \mid \delta(q_2, w) \in F\} = aX_0 \cup bX_2 = aX_0 + bX_2$

Com'è definito l'automa A_2? Devo definire q_2 come stato iniziale, ossia

A_2 = 

A_2 ha due stati (q_0 e q_1) non osservabili

X_2 è il linguaggio riconosciuto da A_2

LINGUAGGI LIBERI DAL CONTESTO

RICORDANDO PRIMA CHE: un linguaggio L è detto acontestuale (o libero dal contesto) se è generato da una grammatica di tipo 2.
RICORDANDO PRIMA CHE: una grammatica G si dice di tipo 2 se ogni regola è della forma $\alpha \to \beta$ dove $\alpha \in M$

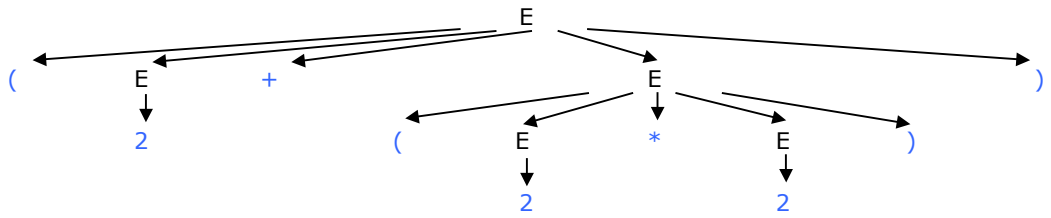
ALBERI DI DERIVAZIONE

Data una grammatica $G = <\Sigma, M, P, S>$ di tipo 2, che genera il linguaggio $L_{(G)}$, un albero di derivazione della parola $w \in L_{(G)}$ in G è un albero ordinato composto da:

- RADICE, è posta centrale in alto, è etichettata con l'assioma della grammatica (S);
- NODI, sono interni all'albero e sono etichettati con metasimboli della grammatica ($m \in M$);
- FOGLIE, sono posti alle estremità inferiori di ciascun ramo, sono etichettati con simboli terminali della grammatica ($\sigma \in \Sigma$) e se letti da sinistra verso destra compongono la parola derivata dalla grammatica;
- ARCHI, sono frecce orientate verso il basso che indicano l'applicazione di una regola di produzione;
    - Se un nodo interno è etichettato con il metasimbolo B e esistono archi che lo uniscono a dei nodi figli etichettati in ordine con i metasimboli $B_1B_2B_3$ questo induce a determinare che esiste in P la seguente regola di produzione: $B \to B_1B_2B_3$.

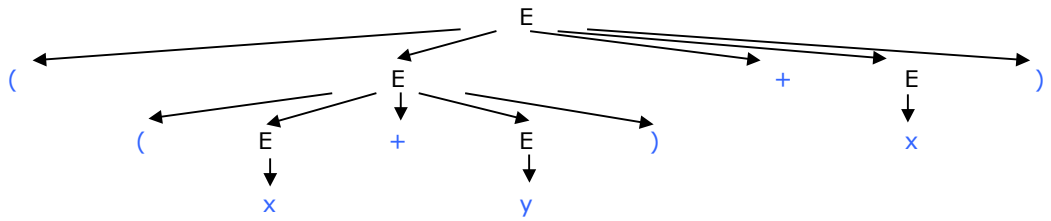
ESEMPIO

Data la grammatica per le espressioni aritmetiche sul numero naturale 2:
$G = <\Sigma = \{(,), +, *, 2\}, M = \{E\}, P = \{E \to (E + E) \mid (E * E) \mid 2\}, E>$
Generiamo la parola w = (2 + (2 * 2)) mediante left most
$E \Rightarrow (E + E) \Rightarrow (2 + E) \Rightarrow (2 + (E * E)) \Rightarrow (2 + (2 * E)) \Rightarrow (2 + (2 * 2))$
Generiamo la parola w = (2 + (2 * 2)) mediante altra derivazione
$E \Rightarrow (E + E) \Rightarrow (E + (E * E)) \Rightarrow (E + (2 * E)) \Rightarrow (E + (2 * 2)) \Rightarrow (2 + (2 * 2))$
Nonostante producano la stessa parola le due derivazioni sono differenti tra loro, quindi possiamo dedurre che ad ogni parola w possono essere associate più derivazioni.
Vediamo ora l'albero di derivazione di w:



ESEMPIO

Data la grammatica per le espressioni aritmetiche sulle incognite x e y:
$G = <\Sigma = \{(,), +, x, y\}, M = \{E\}, P = \{E \to (E + E) \mid x \mid y\}, E>$
Generiamo la parola w = ((x + y) + x) mediante left most
$E \Rightarrow (E + E) \Rightarrow ((E + E) + E) \Rightarrow ((x + E) + E) \Rightarrow ((x + y) + E) \Rightarrow ((x + y) + x)$
L'albero di derivazione corrispondente è:



Si osservi che diverse derivazioni possono avere associato lo stesso albero. Per esempio, riferendoci alla grammatica dell'esercizio, la derivazione seguente derivazione genera lo stesso albero:
$E \Rightarrow (E + E) \Rightarrow (E + x) \Rightarrow ((E + E) + x) \Rightarrow ((E + y) + x) \Rightarrow ((x + y) + x)$

DERIVAZIONE LEFT MOST

Data una grammatica G, diremo che la derivazione è stata effettuata in maniera left most, se si è proceduto nella derivazione sempre partendo dal primo metasimbolo più a sinistra.
Data una grammatica $G = <\Sigma, M, P, S>$ di tipo 2, sia $w \in \Sigma^*$ una parola derivabile dall'assioma S.
La derivazione $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_j \Rightarrow w_{j+1} \Rightarrow \dots \Rightarrow w_m \Rightarrow w$ è detta derivazione più a sinistra (left most) se, per ogni j (con $1 \le j \le m$),
w_{j+1} è ottenuta applicando una regola di produzione al metasimbolo più a sinistra di w_j.
Nei nostri due esempi, la prima derivazione di ciascun esercizio è left most.

DERIVAZIONI EQUIVALENTI

Data una grammatica G, affermeremo che due derivazioni sono equivalenti se hanno associato lo stesso albero di derivazione. Ad una parola w, generata dalla grammatica G, è, in generale, possibile associare diverse derivazioni alle quali, però, è associato uno e un solo albero di derivazione. Si possono considerare tutte le derivazioni associate allo stesso albero come appartenenti ad una stessa classe.

Può essere interessante ottenere un elemento rappresentativo nella classe delle derivazioni equivalenti, compatibili quindi con lo stesso albero. Una soluzione è di considerare, quale rappresentante della classe, la derivazione left most. In questo modo avremmo una corrispondenza biunivoca tra alberi e derivazioni left most. Per ogni albero, infatti, esiste una e una sola derivazione left most e per ogni derivazione left most esiste uno e un solo albero di derivazione.

SIGNIFICATO DI UNA PAROLA

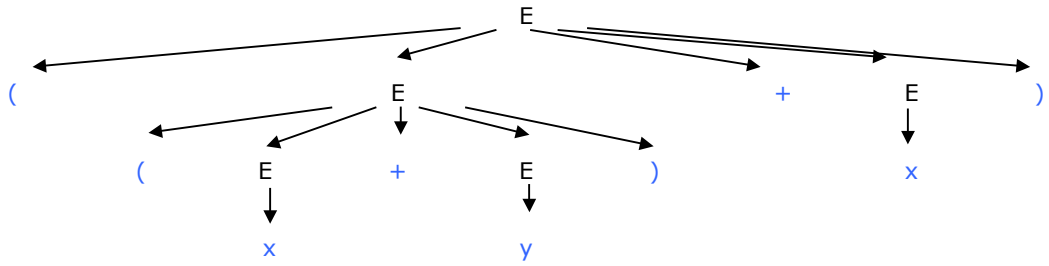
In vari contesti, data una grammatica G di tipo 2 che genera il linguaggio $L_{(G)}$ ed una parola $w \in L_{(G)}$, risulta ragionevole interpretare un albero di derivazione di w in G come significato della parola w.

Una richiesta molto importante, nello studio dei linguaggi, è che ogni parola di L(G) abbia un unico significato.

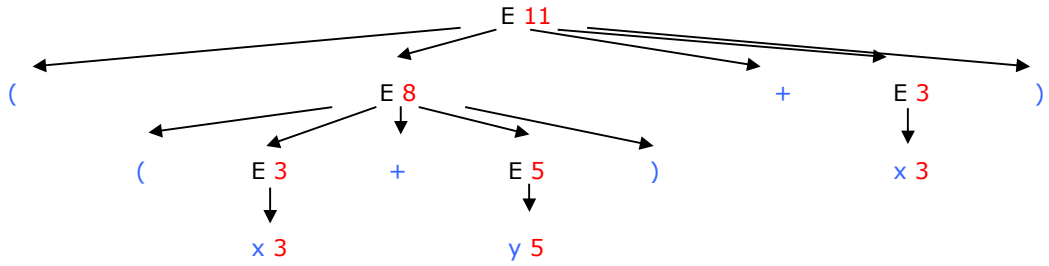
Una parola che rappresenta un'espressione aritmetica ha, per significato il risultato dello svolgimento dell'espressione regolare rappresentata dalla parola stessa.

ESEMPIO DI SIGNIFICATO DI UNA PAROLA

Dato il seguente albero di derivazione, determinare il significato della parola w con x = 3 e y = 5:

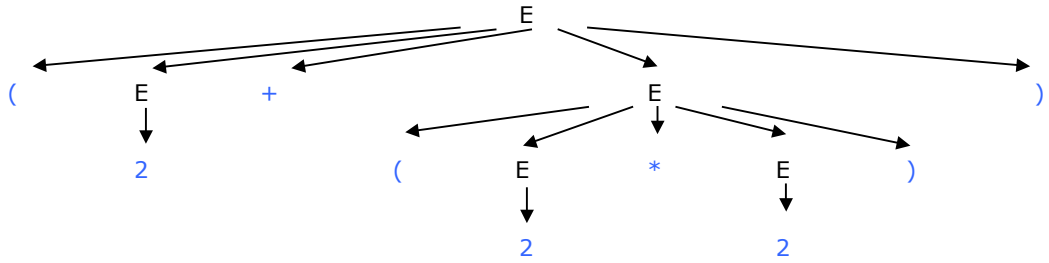


Il significato di ((x + y) + x) è 11, infatti,

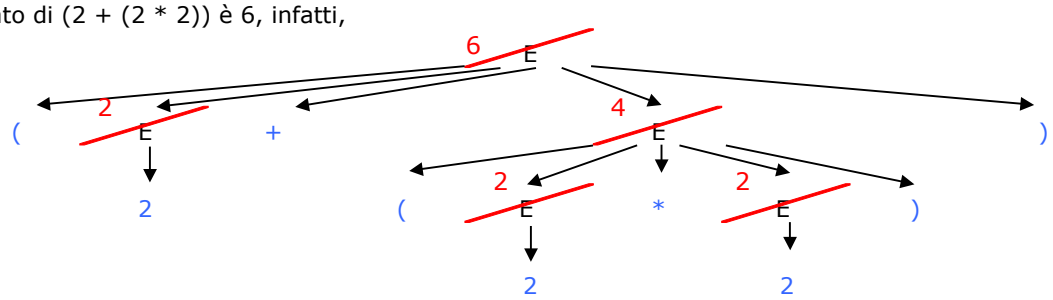


ESEMPIO DI SIGNIFICATO DI UNA PAROLA

Dato il seguente albero di derivazione, determinare il significato della parola w derivante dall'albero:



Il significato di (2 + (2 * 2)) è 6, infatti,



IL PROBLEMA DELL'AMBIGUITÀ

In italiano, ambiguità significa associare ad un termine due o più significati tra loro differenti.

Il problema dell'ambiguità è affrontato per le grammatiche di tipo 2, infatti, per le grammatiche di tipo 3, se esiste ambiguità, la grammatica risulta trasformabile in una grammatica di tipo 3 equivalente priva di ambiguità.

Attribuendo un albero di derivazione di w in G come significato della parola w, se esiste una parola ha diversi alberi di derivazione, e quindi significati diversi, la parola si dice AMBIGUA.

ESEMPIO DI FRASI ITALIANE AMBIGUE

Lei suona il piano e lui la tromba.

Il missionario andava a letto con una vecchia coperta di pelo

ESEMPIO DI PROGRAMMA AMBIGUO

input(x) if(x<0) if(x<0) x = -1*x else x = -1*x output(x)

Questo programma risulta avere due significati, analizziamo le regole di produzioni che ci consentono di generare detto programma, partendo dall'assioma P.

$P \rightarrow$ i C o	# dove C = comando (Metasimbolo), i = input(x), o = output(x)
$C \rightarrow A \mid I \mid E$	# dove A = assegnamento, I = istruzione If, E = istruzione If … Else
$A \rightarrow$ x = -1*x	# effettua l'assegnamento a x di (−1 * x)
$I \rightarrow$ if(T) C	# dove T = test, C = comando, if è simbolo terminale
$E \rightarrow$ if(T) C e C	# dove T = test, C = comando, if ed e = else sono simboli terminali
$T \rightarrow$ x < 0	# test per l'istruzione if

Effettuiamo la derivazione in più passi del programma appena scritto:

P ⇨ i C o

⇨ i I o ⇨ i if(T) C o ⇨ i if(T) E o ⇨ i if(T) if(T) C e C o

⇨ i E o ⇨ i if(T) C e C o ⇨ i if(T) I e C o ⇨ i if(T) if(T) C e C o

Le due derivazioni apparentemente sembrano aver generato lo stesso programma, analizziamo la prima:

Ia deriv: P ⇨ i C o ⇨ i I o ⇨ i if(T) C o ⇨ i if(T) E o ⇨ i if(T) if(T) C e C o

Risulta esser stato generato il seguente programma:

```
input(x)
        if (x<0)
                if (x<0)
                x=-1*x          F(x) = { -x   sse x > 0   se l'input è positivo viene trasformato in negativo
        else                           { opp x = 0
        x=-1*x                         { -x   sse x < 0   entro nel secondo if quindi il numero negativo viene trasformato in un numero positivo
output(x)
```

questo programma quindi calcola la fuzione F(x) = -x

Analizziamo ora il programma generato dalla seconda derivazione:

IIa deriv: P ⇨ i C o ⇨ i E o ⇨ i if(T) C e C o ⇨ i if(T) I e C o ⇨ i if(T) if(T) C e C o

Risulta esser stato generato il seguente programma:

```
input(x)
        if(x<0)
                if (x<0)
                x=-1*x          F(x) = { x    sse x > 0   se l'input è positivo, dato che il primo if non ha else, non viene effettuata alcuna trasformazione
                else                   { opp x = 0
                x=-1*x                 { -x   sse x < 0   entro nel secondo if quindi il numero negativo viene trasformato in un numero positivo
output(x)
```

questo programma quindi calcola la fuzione F(x) = |x|

RICORDANDO PRIMA CHE: ad ogni albero di derivazione è associata una e una sola derivazione left most

RICORDANDO PRIMA CHE: ad ogni albero di derivazione è associato uno e un solo significato della parola

RICORDANDO PRIMA CHE: una parola si dice ambigua se ha diversi significati, ossia se ha almeno due alberi di derivazione o ancora se ha almeno due derivazioni left most differenti

RICORDANDO PRIMA CHE: una parola si dice non ambigua se ha un unico significato, ossia se ha un solo albero di derivazione o ancora se ha un'unica derivazione left most

GRAMMATICA AMBIGUA

Una grammatica G = <Σ, M, P, S> di tipo 2 si dice ambigua se genera almeno una parola ambigua, ossia:

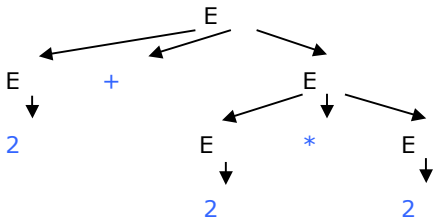
- Se ∃ w ∈ $L_{(G)}$ che ammette due diversi alberi di derivazione ⇒ w è ambigua ⇒ G è ambigua
- Se ∃ w ∈ $L_{(G)}$ che ammette due diverse derivazioni left most ⇒ w è ambigua ⇒ G è ambigua
- Se ∃ w ∈ $L_{(G)}$ che ammette due diversi significati ⇒ w è ambigua ⇒ G è ambigua

ESEMPIO DI GRAMMATICA AMBIGUA

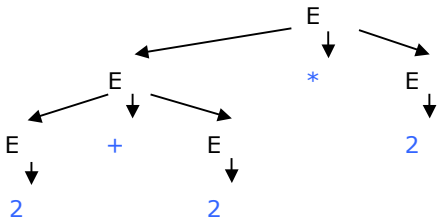
Data la grammatica per le espressioni aritmetiche sul numero naturale 2 non parentesizzate:

G = <Σ = {+, *, 2}, M = {E}, P = {E → E + E | E * E | 2}, E>

Generiamo la parola w = 2 + 2 * 2, essa ammette i seguenti due distinti alberi di derivazione:



il significato di questo albero è 6



il significato di questo albero è 8

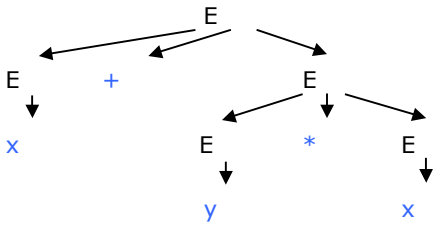
Dato che la grammatica risulta avere almeno una parola ambigua (avente due alberi e quindi due significati), la grammatica G è una grammatica ambigua.

ESEMPIO DI GRAMMATICA AMBIGUA

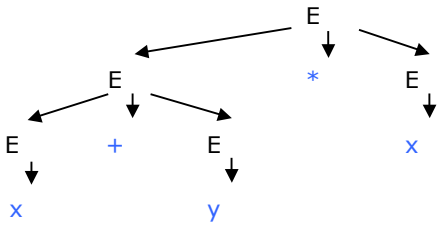
Data la grammatica per le espressioni aritmetiche sul numero naturale 2 non parentesizzate:

G = <Σ = {+, *, x, y}, M = {E}, P = {E → E + E | E * E | x | y}, E>

Generiamo la parola w = x + y * x con x = 3, y = 5,



il significato di questo albero è 18



il significato di questo albero è 24

Dato che la grammatica risulta avere almeno una parola ambigua (avente due alberi e quindi due significati), la grammatica G è una grammatica ambigua.

GRAMMATICA NON AMBIGUA

Una grammatica G = <Σ, M, P, S> di tipo 2 si dice non ambigua se genera tutte e sole parole non ambigue,

- Se ∀ w ∈ $L_{(G)}$ ∃! Albero di derivazione ⇒ w non è ambigua ⇒ G non è ambigua
- Se ∀ w ∈ $L_{(G)}$ ∃! derivazione left most ⇒ w non è ambigua ⇒ G non è ambigua
- Se ∀ w ∈ $L_{(G)}$ ∃! significato ⇒ w non è ambigua ⇒ G non è ambigua

ESEMPIO DI GRAMMATICA NON AMBIGUA

Data la grammatica per le espressioni aritmetiche parentesizzate:

G = <Σ = {(,), +, *, x, y}, M = {E}, P = {E → (E + E) | (E * E) | x | y}, E>

È possibile dimostrare che ogni parola del linguaggio generato ammette esattamente un albero di derivazione, ossia un'unica derivazione left most. La dimostrazione è effettuata per induzione sulla lunghezza della parola:

- Le parole di lunghezza 1 in $L_{(G)}$ sono x e y ed hanno un'unica derivazione left most;
- Sia w ∈ $L_{(G)}$ | |w|>1, essa sarà della forma (w_1+w_2) o (w_1*w_2), dove |w_1| < n e |w_2| < n. Possiamo supporre, senza perdere di generalità, che w=(w_1+w_2). Allora w è generata da una derivazione "left most" del tipo E⇒(E+E) ⇒ (w_1+E) ⇒ (w_1+w_2). Per ipotesi di induzione, esiste un'unica derivazione left most del tipo E ⇒ w_1 e un'unica left most del tipo E ⇒ w_2, quindi esiste un'unica derivazione left most di w.

ESEMPIO DI GRAMMATICA NON AMBIGUA

Data la grammatica per le espressioni aritmetiche sul numero naturale 2 parentesizzate:

G = <Σ = {(,), +, *, 2}, M = {E}, P = {E → (E + E) | (E * E) | 2}, E>

È possibile dimostrare che G non è ambigua semplicemente basandosi sul fatto che le parentesi impongono una precedenza e, quindi, non è possibile effettuare più di una derivazione differente.

li

RICORDANDO PRIMA CHE: se L è generato da una grammatica G di tipo 3 ambigua, è sempre possibile costruire una grammatica G' di tipo 3 (equivalente a G) non ambigua per generare L.
È, infatti, sufficiente costruire l'automa deterministico che riconosce L, la grammatica G relativa al DFA è una grammatica di tipo non ambiguo.

LINGUAGGIO INERENTEMENTE AMBIGUO

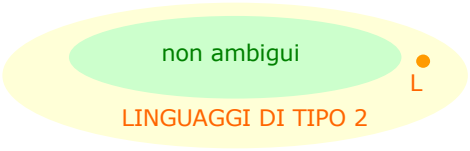
Un linguaggio acontestuale L si dice inerentemente ambiguo se ogni grammatica di tipo 2 che lo genera è ambigua, ossia se non è possibile trovare una grammatica equivalente, di tipo 2, non ambigua, che generi L.

DOMANDA

Un linguaggio di programmazione è un linguaggio inerentemente ambiguo o non ambiguo?

I linguaggi di programmazione sono non ambigui, infatti, è possibile progettare dei compilatori che o lavorano con linguaggi ambigui e sono in grado di riconoscere quale dei significati multipli è realmente associato alla parola, oppure lavorano con linguaggi non ambigui, in ogni caso un linguaggio di programmazione ambiguo può sempre essere disambiguato mediante l'utilizzo del compilatore.

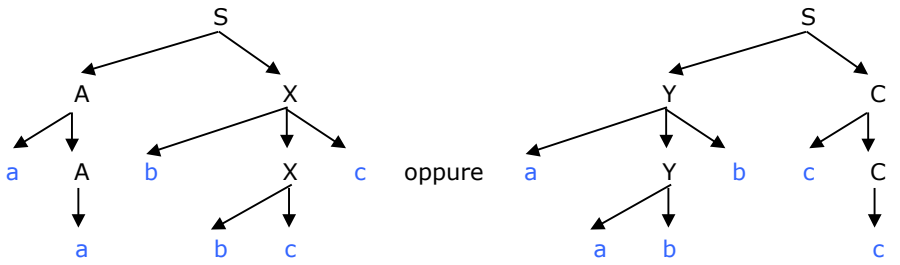
LINGUAGGI NON AMBIGUI $\subseteq$ LINGUAGGI TIPO 2



Definiamo L = $\{a^m b^n c^k \mid$ o m = n oppure n = k$\}$
Costruiamo ora la grammatica G = $<\Sigma, M, P, S>$, in grado di generare L:

- Alfabeto terminale $\Sigma = \{a, b, c\}$
- Alfabeto non terminale M = { S, X, C, A, Y }
- P = {
  1) $S \rightarrow YC$
  2) $S \rightarrow AX$
  3) $Y \rightarrow aYb$
  4) $Y \rightarrow ab$
  5) $X \rightarrow bXc$
  6) $X \rightarrow bc$
  7) $A \rightarrow aA$
  8) $A \rightarrow a$
  9) $C \rightarrow cC$
  10) $C \rightarrow c$
- Assioma S

G = $<\Sigma=\{a, b, c\}$, M={S, X, C, A, Y}, P={$S \rightarrow YC|AX$, $Y \rightarrow aYb|ab$, $X \rightarrow bXc|bc$, $A \rightarrow aA|a$, $C \rightarrow cC|c$}, S>
Detto linguaggio è acontestuale, infatti, è generato da una grammatica di tipo 2.
La grammatica G è ambigua in quando le parole del tipo $a^j b^j c^j$ ammettono due distinti alberi di derivazione.
Per esempio $a^2b^2c^2$ può essere ottenuto come segue:



La grammatica G è ambigua in quando le parole del tipo $a^j b^j c^j$ ammettono due distinte derivazioni left most.
Per esempio $a^2b^2c^2$ può essere ottenuto come segue:
$S \Rightarrow^2 \underline{A}X \Rightarrow^7 a\underline{A}X \Rightarrow^8 aa\underline{X} \Rightarrow^5 aab\underline{X}c \Rightarrow^6 aabbcc$
Oppure
$S \Rightarrow^1 \underline{Y}C \Rightarrow^3 a\underline{Y}bC \Rightarrow^4 aabb\underline{C} \Rightarrow^9 aabbc\underline{C} \Rightarrow^{10} aabbcc$

FORME NORMALI DI CHOMSKY E GREIBACH

Per far vedere che ogni grammatica di tipo 2 ammette la forma normale di Chomsky e di Greibach, per semplicità, ammetteremo solo linguaggi generati da grammatiche G di tipo 2 non contenenti la parola vuota.
Cos'è un linguaggio non contenente la parola vuota?
Se L è un linguaggio di tipo 2, contenente ε, ossia un linguaggio generato da una grammatica con regole di produzione della forma $\alpha \to \beta$ con $\alpha \in M$ e $\beta \in (M \cup \Sigma)^*$. Si può affermare che L = L' $\cup$ {ε}, dove L' è un linguaggio generato da una grammatica con regole di produzione della forma $\alpha \to \beta$ con $\alpha \in M$ e $\beta \in (M \cup \Sigma)^+$.
Per ogni linguaggio L libero da contesto, non contenente la parola vuota, esiste una grammatica G_{FNC} in forma normale di Chomsky che genera L ed esiste una grammatica G_{FNG} in forma normale di Greibach che genera L.

FORMA NORMALE DI GREIBACH (FNG)

Una grammatica G = <Σ, M, P, S> di tipo 2 si dice in forma normale di Greibach e si scrive G_{FNG} se le regole di produzione sono in una delle seguenti forme:

$A \to \sigma W$ # con $A \in M$ $\sigma \in \Sigma$ $W \in M^*$

Chiaramente per poter trasformare le regole di produzione in FNG si necessita di un algoritmo in grado di trasformare una regola della forma $A \to B$ in una regola della forma $A \to \sigma W$, dove W può anche essere la parola vuota. Noi non vedremo l'algoritmo di trasformazione.

ESEMPIO DI TRASFORMAZIONE IN FNG

Data la grammatica per le espressioni aritmetiche parentesizzate:
G = <Σ = {(,), +, *, x, y}, M = {E}, P = {E → (E + E) | (E * E) | x | y}, E>
Analizziamo le regole di produzione di cui si compone la grammatica e trasformiamole in regole in FNG:

- E → x # è della forma $A \to \sigma W$ con σ = x, W = ε ok per FNG
- E → y # è della forma $A \to \sigma W$ con σ = y, W = ε ok per FNG
- E → (E * E) # non è della forma $A \to \sigma W$, si necessita di trasformarla:
  - Introduciamo due nuovi metasimboli (G, H) e le seguenti regole di produzione:
    - G → * # è della forma $A \to \sigma W$ con σ = *, W = ε ok per FNG
    - H →) # è della forma $A \to \sigma W$ con σ =), W = ε ok per FNG
  - Ottengo quindi E → (EGEH # è della forma $A \to \sigma W$ con σ = (, W = EGEH ok per FNG
- E → (E + E) # non è della forma $A \to \sigma W$, si necessita di trasformarla:
  - Introduciamo un nuovo metasimbolo (P) e la seguente regola di produzione:
    - P → + # è della forma $A \to \sigma W$ con σ = +, W = ε ok per FNG
  - Ottengo quindi E → (EPEH # è della forma $A \to \sigma W$ con σ = (, W = EPEH ok per FNG

G_{FNG} = <Σ_{FNG}=Σ, M_{FNG} = {E, G, H, P}, P_{FNC} = {E → x | y | (EGEH | (EPEH, G → *, H →), P → +}, E>

FORMA NORMALE DI CHOMSKY (FNC)

Una grammatica G = <Σ, M, P, S> di tipo 2 si dice in forma normale di Chomsky e si scrive G_{FNC} se le regole di produzione sono in una delle seguenti forme:

$$\begin{cases} A \to BC & \text{\# con A, B, C} \in M \\ A \to \sigma & \text{\# con } \sigma \in \Sigma \end{cases}$$

Chiaramente per poter trasformare le regole di produzione in FNC si necessita di un algoritmo in grado di trasformare una regola della forma $A \to B$ in una regola della forma $A \to BC \mid A \to \sigma$. Noi non vedremo l'algoritmo di trasformazione.

ESEMPIO DI TRASFORMAZIONE IN FNC

Data la grammatica che genera il linguaggio $L = a^n b^n$:
G = <Σ = {a, b}, M = {S}, P = {S → aSb | ab}, S>
Analizziamo le regole di produzione di cui si compone la grammatica e trasformiamole in regole in FNC:

- S → ab # non è né della forma $A \to \sigma$ né della forma $A \to BC$, si necessita di trasformarla:
  - Introduciamo due nuovi metasimboli (A, B) e le seguenti regole di produzione:
    - A → a # è della forma $A \to \sigma$ con σ = a ok per FNC
    - B → b # è della forma $A \to \sigma$ con σ = b ok per FNC
  - Ottengo quindi S → AB # è della forma $A \to BC$ con B = A e C = B ok per FNC
- S → aSb # non è né della forma $A \to \sigma$ né della forma $A \to BC$, si necessita di trasformarla:
  - Utilizziamo i metasimboli A e B con rispettive regole di produzione,
    ottengo quindi S → ASB # non è né della forma $A \to \sigma$ né $A \to BC$, si necessita di ridurla:

    S → A S B
    (A S) = C

    - Introduciamo un nuovo metasimbolo (C) e la seguente regola di produzione:
      - C → AS # è della forma $A \to BC$ con B = A e C = S ok per FNC
  - Ottengo quindi S → CB # è della forma $A \to BC$ con B = C e C = B ok per FNC

G_{FNC} = <Σ_{FNC}=Σ, M_{FNC} = {S, A, B, C}, P_{FNC} = {S → AB | CB, A → a, B → b, C → AS}, S>
Abbiamo trasformato G in G_{FNC} dove quest'ultima risulta essere in forma normale di Chomsky.

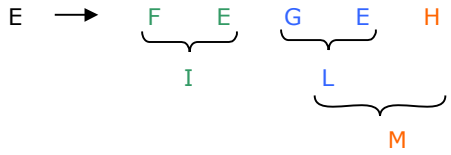
ESEMPIO DI TRASFORMAZIONE IN FNC

Data la grammatica per le espressioni aritmetiche parentesizzate:

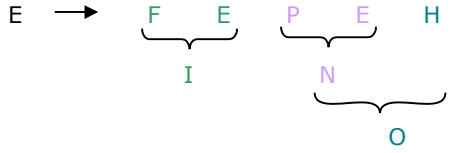
G = < Σ = {(,), +, *, x, y},
M = {E},
P = { E → (E + E)
E → (E * E)
E → x
E → y},
E>

Analizziamo le regole di produzione di cui si compone la grammatica e trasformiamole in regole in FNC:

- E → x # è della forma A → σ con σ = x ok per FNC
- E → y # è della forma A → σ con σ = y ok per FNC
- E → (E * E)# non è né della forma A → σ né della forma A → BC, si necessita di trasformarla:
  - Introduciamo tre nuovi metasimboli (F, G, H) e le seguenti regole di produzione:
    - F → (# è della forma A → σ con σ = (ok per FNC
    - G → * # è della forma A → σ con σ = * ok per FNC
    - H →) # è della forma A → σ con σ =) ok per FNC
  - Ottengo quindi E → FEGEH # non è né della forma A → σ né A → BC, si necessita di ridurla:



    - Introduciamo tre nuovi metasimboli (I, L, M) e le seguenti regole di produzione:
      - I → FE # è della forma A → BC con B = F e C = E ok per FNC
      - L → GE # è della forma A → BC con B = G e C = E ok per FNC
      - M → LH # è della forma A → BC con B = L e C = H ok per FNC
  - Ottengo quindi E → IM # è della forma A → BC con B = I e C = M ok per FNC
- E → (E + E) # non è né della forma A → σ né della forma A → BC, si necessita di trasformarla:
  - Utilizziamo i metasimboli F e H con rispettive regole e introduciamo il nuovo metasimbolo P e la seguente regola di produzione:
    - P → + # è della forma A → σ con σ = + ok per FNC
  - Ottengo quindi E → FEPEH # non è né della forma A → σ né A → BC, si necessita di ridurla:

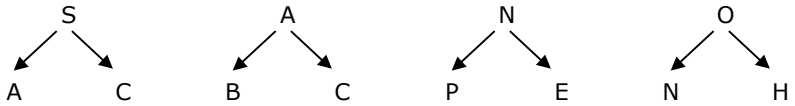


    - Introduciamo due nuovi metasimboli (N, O) e le seguenti regole di produzione:
      - N → PE # è della forma A → BC con B = P e C = E ok per FNC
      - O → NH # è della forma A → BC con B = N e C = H ok per FNC
  - Ottengo quindi E → IO # è della forma A → BC con B = I e C = O ok per FNC

G' = <Σ = {(,), +, *, x, y},
M = {E, F, G, H, I, L, M, N, O, P},
P = { E → IM
E → IO
E → x
E → y
F → (
G → *
H →)
P → +
I → FE
L → GE
M → LH
N → PE
O → NH },
E>

ALBERI BINARI DI FNC

Gli alberi di derivazione, per le parole generate da grammatiche in forma normale di Chomsky, hanno la particolarità, escludendo l'ultimo livello (livello che permette la creazione delle foglie), di essere alberi binari.
Per albero binario si intende un particolare tipo di albero di derivazione dove ogni nodo ha sono ed esclusivamente 2 figli, sono un esempio di nodi binari:



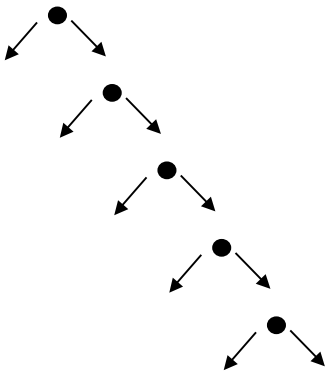
Perché si necessita di escludere l'ultimo livello? Perché l'ultimo livello è un livello unario, infatti, all'ultimo livello avvengono trasformazioni del tipo $A \rightarrow \sigma$ le quali consentono di trasformare i metasimboli in simboli terminali.

ALTEZZA DEGLI ALBERI BINARI

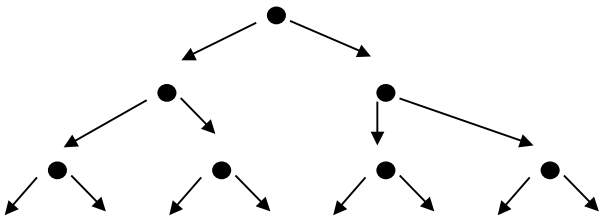
L'altezza di un albero binario è il numero massimo di rami che devo seguire per raggiungere le foglie, partendo dall'assioma (radice dell'albero).
Il numero di foglie di cui l'albero si compone mi fornisce informazioni sull'altezza dell'albero stesso a seconda che quest'ultimo risulti avere una struttura a catena o una struttura bilanciata.
Se l'albero ha forma a catena $\Rightarrow$ altezza = # foglie
Se l'albero ha forma bilanciata $\Rightarrow$ altezza = $\log_2$#foglie

FORMA A CATENA E FORMA BILACIATA

forma a catena



forma bilanciata



Nella forma a catena:
altezza = # foglie

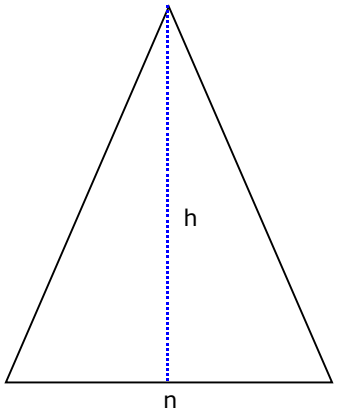
nella forma bilanciata:
altezza = $\log_2$#foglie

Se ne deduce quindi che l'altezza di un albero binario che un valore che può oscillare tra due valori precisi:

$$\log_2\#\text{foglie} < \text{altezza} < \#\text{foglie}$$

PROPRIETÀ SUGLI ALBERI BINARI

Sia n il numero di foglie di un albero binari
Sia h l'altezza di un albero binario



$\log_2 n \leq h < n$ $\quad h \in [\log_2 n, n)$
$h < n \leq 2^h$ $\quad n \in (h, 2^h]$

PUMPING LEMMA

Il pumping lemma è una condizione necessaria ma non sufficiente per determinare se un linguaggio è di tipo 2. Questo induce a definire il pumping lemma come lo strumento in grado di dimostrare che, un linguaggio L, non è di tipo 2.

CONDIZIONE NECESSARIA

Se un linguaggio L non soddisfa il pumping lemma allora sicuramente L non è di tipo 2

CONDIZIONE NON SUFFICIENTE

Se un linguaggio L soddisfa il pumping lemma questo non è sufficiente per definire L come linguaggio di tipo 2

ENUNCIATO DEL PUMPING LEMMA

Per ogni linguaggio L, libero dal contesto (di tipo 2), esiste una costante H > 0 tale che, ogni parola $z \in L$, con $|z| > H$, è scomponibile in un particolare prodotto di concatenazione così definito:

Se L tipo 2

$\exists H > 0 \mid$ se $z \in L$ e $|z| > H \Rightarrow z = u \cdot v \cdot w \cdot x \cdot y$

Ossia z risulta scritta come prodotto tra i fattori u, v, w, x, y

CONDIZIONI PER LA DECOMPOSIZIONE: nel decomporre z come prodotto dei 5 fattori sopra elencati bisogna tener conto delle seguenti 3 condizioni:

1. $|vx| \geq 1$
2. $|vwx| \leq H$
3. $\forall k \geq 0$ vale che $uv^kwx^ky \in L$

Vediamo ora in dettaglio le tre condizioni:

$|vx| \geq 1$

La sottoparola vx, ovvero quella parola composta dalla concatenazione del fattore v con il fattore x deve avere una lunghezza strettamente maggiore di zero, ossia la lunghezza $|vx| = |v| + |x| > 0$.

Questo significa che v e x non possono essere contemporaneamente la parola vuota, infatti, se fossero contemporaneamente $v = x = \varepsilon$ si avrebbe

$|vx| = |v| + |x| = |\varepsilon| + |\varepsilon| = 0 + 0 = 0$ dove 0 non è maggiore di zero.

$|vwx| \leq H$

Il nucleo centrale della scomposizione di z (ossia il fattore vwx) non deve avere lunghezza superiore alla costante H di riferimento.

$\forall k \geq 0$ vale che $uv^kwx^ky \in L$

Preso un $k \geq 0$ (dove $k \in N$, i numeri naturali) la parola che si ottiene effettuando la concatenazione di u, v, w, x, y dove, però, i fattori v e x sono presi alla potenza k^{-esima}, detta parola deve appartenere al linguaggio.

$\forall k \geq 0$ $\quad u \cdot v \cdot v \cdot \ldots \cdot v \cdot w \cdot x \cdot x \cdot \ldots \cdot x \cdot y \in L$ (con v e x ripetuti k volte)

DIMOSTRAZIONE DEL PUMPING LEMMA

Ora si necessita di dimostrare i 3 punti di cui si compone l'enunciato e l'esistenza di H. Per dimostrare il pumping lemma proviamo a leggere l'enunciato pezzo per pezzo:

PER OGNI LINGUAGGIO L, LIBERO DAL CONTESTO (DI TIPO 2), ESISTE UNA COSTANTE H > 0

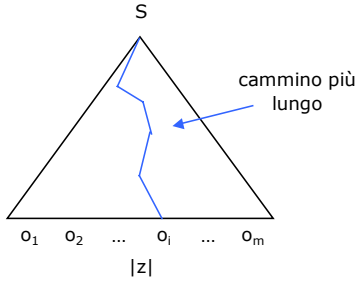
Se L è un linguaggio di tipo 2 allora $\exists$ G di tipo 2 che lo genera. Dato che ogni grammatica di tipo 2 è trasformabile nella forma normale desiderata, in questo caso scegliamo di trasformare G in G_{FNC} (grammatica in forma normale di Chomsky) detta scelta è dettata dal fatto che, se una grammatica è in FNC ad essa si possono applicare le proprietà sugli alberi binari di Chomsky.

L'esistenza di una costante H è facilmente dimostrabile mediante il seguente assegnamento: $H = 2^{h+1}$ dove H è la costante dell'enunciato e h è il numero di metasimboli della grammatica G_{FNC} (in forma normale di Chomsky), ossia dato M l'insieme dei metasimboli, allora h = cardinalità di M.

Con tale operazione abbiamo garantito l'esistenza di H.

OGNI PAROLA $z \in L$, CON $|z| > H$

Cosa significa $z \in L$? significa che z è una parola generata dalla grammatica G_{FNC} (in forma normale di Chomsky), ossia che per z esiste un albero di derivazione binario dove, sulle foglie dell'albero, compare proprio la parola z, l'imposizione che $|z| > H$ significa che il numero di foglie dell'albero binario è maggiore della costante H precedentemente definita. Avrò quindi il seguente albero:



RICORDANDO PRIMA CHE: L'altezza di un albero binario è il numero massimo di rami che devo seguire per raggiungere le foglie, partendo dall'assioma (radice dell'albero), ossia l'altezza di un albero è definita dal cammino più lungo all'interno dell'albero.

L'altezza di un albero binario è il numero massimo di rami che devo seguire per raggiungere le foglie, partendo dall'assioma (radice dell'albero).
Sulle foglie dell'albero binario ho i simboli $\sigma_1\ \sigma_2 \dots \sigma_i \dots \sigma_m$ che compongono la parola z, ossia |z| = #foglie
So che il numero di foglie dell'albero è maggiore di H, questo è garantito da |z| > H
Se |z| > H
$\Rightarrow$ # foglie > H
E |z| = # foglie

RICORDANDO PRIMA CHE: siano a, b $\in$ N | a $\geq$ 1, b $\geq$ 1 (a, b numeri naturali strettamente positivi)
se a > b $\Rightarrow$ log a > log b
se b > a $\Rightarrow$ log b > log a
se a = b $\Rightarrow$ log a = log b
se $a = 2^k \Rightarrow \log_2 a = k$

Supponiamo che il cammino più lungo dell'albero ricada sulla foglia σ_i, dato che sappiamo che l'altezza dell'albero risulta essere una quantità almeno $\geq \log_2$#foglie otteniamo la seguente espressione:

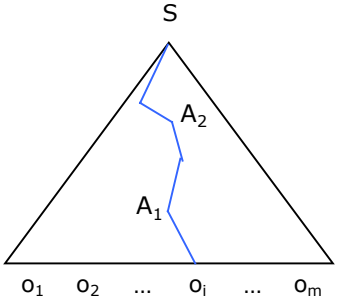
Altezza $\geq \log_2$#foglie	ma # foglie = |z|
$\Rightarrow$ Altezza $\geq \log_2\|z\|$	ma |z| > H, dato che H e |z| > 1 $\Rightarrow \log_2\|z\| > \log_2 H$
$\Rightarrow$ Altezza $\geq \log_2\|z\| > \log_2 H$	ma $H = 2^{h+1}$
$\Rightarrow$ Altezza $\geq \log_2\|z\| > \log_2 H = \log_2 2^{h+1}$	ma $\log_2 2^{h+1} = h+1$
$\Rightarrow$ Altezza $\geq \log_2\|z\| > h+1$	ma se a $\geq$ b > c $\Rightarrow$ a > c
$\Rightarrow$ Altezza $\geq h+1$	

Ma che cos'è h? E' il numero di metasimboli della grammatica;
Questo che cosa ci fa concludere? Che nel percorso da S a σ_i vi sarà una ripetizione in più nodi del medesimo metasimbolo.
Perché è prevista una ripetizione? La risposta è fornita dal principio della piccionaia:

PRINCIPIO DELLA PICCIONAIA:
sia 10 (h+1) il numero di piccioni che devono occupare le cellette della piccionaia;
sia 9 (h) il numero di cellette di cui la piccionaia si compone;
anche prevedendo di disporre un piccione in ogni celletta, quando avrò riempito tutte le cellette mi avanzerà comunque un piccione, se devo riporre obbligatoriamente tutti i piccioni nelle cellette ciò implicherà che il 10° piccione dovrà esser posto in una celletta già occupata.

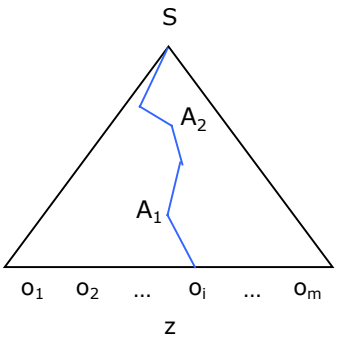
Analogamente al principio della piccionaia noi possediamo h metasimboli e almeno h+1 nodi sul percorso più lungo, è chiaro che volendo distribuire tutti i metasimboli sui nodi, sarò, in ogni caso, costretto a ripetere almeno un metasimbolo su due nodi distinti.
Come faccio a trovare il metasimbolo ripetuto nel percorso da S a σ_i?
Analizzo il percorso da σi verso S (ossia dalla foglia verso la radice, dal simbolo terminare verso l'assioma), tutti i nodi che incontro saranno etichettati con A_i, ossia

- Parto da σ_i
- Salgo e il primo metasimbolo che incontro lo etichetto con A_1
- Salgo da A_1 in direzione S e il primo metasimbolo che incontro lo etichetto con A_2 (secondo metasimbolo partendo da σ_i)
- Salgo da A_2 in direzione S e il primo metasimbolo che incontro lo etichetto con A_3 (terzo metasimbolo partendo da σ_i)
- Proseguo fino a S con questa tecnica

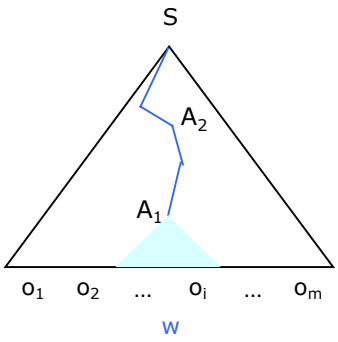


È SCOMPONIBILE IN UN PARTICOLARE PRODOTTO DI CONCATENAZIONE COSÌ DEFINITO:
SE L TIPO 2, $\exists$ H > 0 | SE z $\in$ L e |z| > H $\Rightarrow$ z = u · v · w · x · y
OSSIA z RISULTA SCRITTA COME PRODOTTO TRA I FATTORI u, v, w, x, y

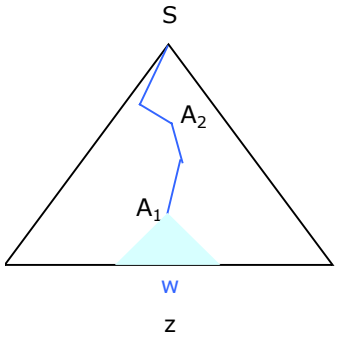
Vediamo come si ottiene la scomposizione in fattori mediante l'utilizzo dell'albero:



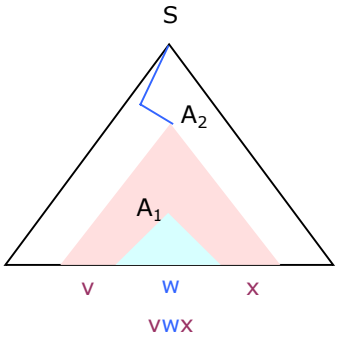
indico il sottoalbero dell'albero di derivazione generato da A_1, ossia:



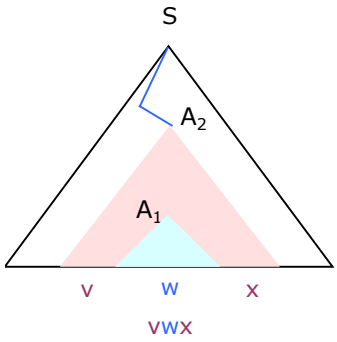
Adesso ho trovato quel fattore della parola di z che ha come radice A_1 cui assegno il nome di w.



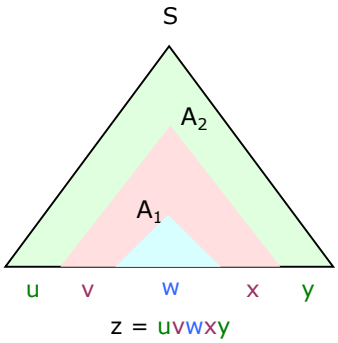
indico il sottoalbero dell'albero di derivazione generato da A_2, ossia:



Sono riuscita in questo modo a trovare altri due fattori di z ossia il fattore v (è il fattore avente radice A_2) che si trova a sinistra del fattore w, e il fattore x (è il fattore avente radice A_2) che si trova a destra del fattore w.



indico l'albero di derivazione generato da S, ossia:

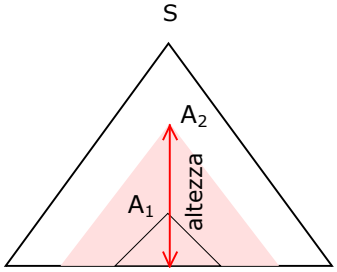


Sono riuscito a dimostrare che z è scomponibile come prodotto dei fattori u, v, w, x, y.

|vx| > 1

Se A_1 coincide con A_2 allora v e x sarebbero entrambe, contemporaneamente, vuote (ossia uguali ad ε).
Noi ragioneremo sul fatto che A_1 è un nodo che non coincide con A_2 (infatti li supponiamo distinti).
Questo mi garantisce che almeno uno tra "v" e "x" deve essere diverso da ε.

|vwx| < H

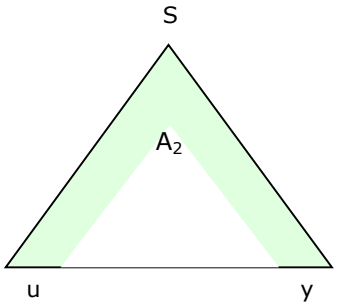


L'altezza dell'albero con radice A_2 è sicuramente minore o al più uguale ad h+1 in quanto, partendo dal fondo ho la garanzia che entro h+1 nodi trovo una ripetizione (principio della piccionaia), essendo l'altezza di A_2 sicuramente minore di h+1 so che il numero di foglie del sottoalbero con radice A_2 sarà un valore compreso tra il logaritmo in base due di h+1 e 2 elevato alla h+1.
Altezza $\leq$ h+1 $\Rightarrow$ $\log_2 h+1 <$ # foglie $\leq 2^{h+1}$

Sto quindi dicendo che il numero di foglie di A_2 è sicuramente $\leq H$, ossia, dato che le foglie di A_2 sono "vwx" sto garantendo che la lunghezza del fattore vwx sia minore o al più uguale alla lunghezza della costante H, infatti, $|vwx| \leq H$

$\forall\ k \geq 0\ \ u\ v^k w x^k\ y \in L$

Per verificare detta condizione si necessita di analizzare l'albero di derivazione, tagliandolo a pezzettini, come se fosse un puzzle.



Questa porzione di albero mi fornisce la seguente informazione:
Patendo dall'assioma S, applicando più regole di produzione è possibile ottenere a seguente forma sentenziale uA_2y, ossia esiste $S \Rightarrow^* uA_2y$

LINGUAGGIO: capacità d'uso e uso stesso di un qualunque sistema di simboli atto a comunicare

un linguaggio può essere classificato in modi diversi a seconda della caratteristica secondo la quale lo si studia:

- A seconda se è trattabile interamente in maniera automatica (mediante l'utilizzo di macchine):
  - Se è trattabile in maniera automatica prende il nome di formale ossia definito mediante regole rigorose e precise
  - Se non è trattabile in maniera automatica prendono il nome di non formale
- A seconda della cardinalità:
  - Se è un linguaggio a cardinalità finita (con un numero finito di elementi) prende il nome di linguaggio finito
    - un linguaggio finito viene rappresentato in maniera estensiva ossia mediante l'elencazione di tutti gli elementi di cui si compone; esempio di linguaggio finito è il linguaggio vuoto (∅) avente cardinalità zero, un altro esempio è il linguaggio contenente solo la parola vuota ({ε}) avente cardinalità 1.
  - Se è un linguaggio a cardinalità infinita (con un numero infinito di elementi) prende il nome di linguaggio infinito
    - un linguaggio infinito, non potendo esser rappresentato in maniera estensiva, richiede una rappresentazione intensiva, si necessita quindi di definire una proprietà (descritta con una quantità finita di informazioni) che risulti vera su tutti e soli gli elementi del linguaggio.
- A seconda delle possibili decomposizioni delle parole che appartengono a L^+
  - Se tutte le parole che appartengono a L^+ sono decomponibili in maniera univoca allora il linguaggio prende il nome di codice
    - Inoltre se ogni parola di L non è prefisso delle altre parole di L allora il codice prende il nome di codice prefisso
  - Se esiste almeno una parola che appartiene ad L^+ decomponibile in più modi allora il linguaggio prende il nome di non codice
- A seconda che il programma W che specifica il linguaggio implementa una procedura o un algoritmo
  - Se il programma W che specifica il linguaggio L implementa un algoritmo allora L prende il nome di linguaggio ricorsivo, l'algoritmo restituirà 1 se la parola appartiene a L e 0 se la parola non appartiene a L.
  - Se il programma W che specifica il linguaggio L implementa una procedura allora L prende il nome di linguaggio ricorsivamente numerabile, la procedura restituirà 1 se la parola appartiene a L e non terminerà (entrerà in loop) se la parola non appartiene a L.
- A seconda che il linguaggio ammetta o non ammetta un calcolo logico
  - Se L ammette un calcolo logico allora il linguaggio è ricorsivamente numerabile
  - Se L non ammette calcolo logico allora il linguaggio non è ricorsivamente numerabile
- A seconda della grammatica che genera il linguaggio
  - Se L ammette una grammatica che lo genera allora L è ricorsivamente numerabile
    - Se L ammette una grammatica di tipo 0 che lo genera allora L è un linguaggio di tipo 0
    - Se L ammette una grammatica di tipo 1 che lo genera allora L è un linguaggio di tipo 1, anche detto linguaggio contestuale (dipendente da contesto)
    - Se L ammette una grammatica di tipo 2 che lo genera allora L è un linguaggio di tipo 2, anche detto linguaggio acontestuale (libero da contesto)
    - Se L ammette una grammatica di tipo 3 che lo genera allora L è un linguaggio di tipo 3, anche detto linguaggio regolare
    - Esiste un teorema, detto teorema dell'inclusione che definisce la seguente relazione tra i linguaggi $R3 \subset R2 \subset R1 \subset R0$
- A seconda che il linguaggio sia riconosciuto da un automa
  - Se L è definito da un automa a stati finiti allora L è un linguaggio regolare

Sui linguaggi è possibile operare mediante le operazioni: unione, intersezione, complemento, prodotto (il quale consiste nel effettuare il prodotto di giustapposizione tra tutte le parole del primo linguaggio con tutte le parole del secondo linguaggio), potenza e chiusura di kleene.