Lezione dell’1 Marzo 2022

Docente: Palano Beatrice

homepage: [palano.di.unimi.it](http://palano.di.unimi.it)

email: [palano@unimi.it](mailto:palano@unimi.it)

Ariel

Sugli avvisi in bacheca abbiamo il link per le lezioni del martedì e del giovedì. Le slide e le registrazioni le troviamo in “Contenuti > Lezioni e materiali didattici”.

Appunti in formato PDF e lezioni dell’anno scorso. Abbiamo anche la sezione “Esercizi svolti” con 3 temi d’esame svolti.

Su “Informazioni sul corso” abbiamo gli appelli e le modalità d’esame (no parziali).

Com’è fatto l’esame?

Prova scritta + prova orale (raramente, a discrezione del docente).

Allo scritto potranno esserci domande aperte di teoria.

Per accedere all’esame bisogna essere iscritti al SIFA (chiedere alla segreteria didattica).

Su Ariel abbiamo il programma totale e lezione per lezione. Sotto “informazioni del corso” abbiamo anche le dispense del corso.

Nel corso vedremo l’informatica dal punto di vista teorico

Programma del corso

Linguaggi formali (grammatiche) e automi

Linguaggio: insieme di frasi per la comunicazione tra entità diverse

Esempi:

* Linguaggio naturale (italiano, inglese, ecc…) per la comunicazione tra le persone ;
* Linguaggio di programmazione (Java, C, C++, Python, Go) per la comunicazione tra uomo e macchina ;
* Codice morse (codifica di una parole con punti e linee) .

Per conoscere un linguaggio ho bisogno di:

* Vocaboli o parole ;
* Sintassi, cioè le regole per costruire le frasi o i programmi .

Formali: considero questi concetti in maniera precisa utilizzando la matematica.

Vantaggio: uso di dispositivi elettronici per l’automatizzazione delle operazioni.

Esempi positivi:

* compilatori = traduzione dei programmi con la creazione di un eseguibile ;
* interpreti = per l’esecuzione dei programmi prendendo i programmi e mandandoli in esecuzione .

Esempi negativi:

* traduzione di linguaggi naturali, molto sconveniente dato che esso può essere diviso in linguaggio parlato (rappresentato da un segnale continuo) e in linguaggio scritto (con caratteri divisi da spaziatura).

Il linguaggio naturale è molto vasto, ma se prendiamo sottolinguaggi (esempio, messaggi delle stazioni ferroviarie) diventa più semplice analizzarli.

Per l’intero linguaggio abbiamo un problema di ambiguità (ad esempio pèsca e pésca).

Per trattarlo dobbiamo risolvere questo problema.

Per specificare un linguaggio ho bisogno di un sistema formale:

* Sistemi generativi, formati da vocaboli e regole che producono le “grammatiche” ;
* Sistemi riconoscitivi, formati da macchine a stati finiti che producono gli “automi” .

Altra visione

Possiamo pensare a un linguaggio come un problema:

Linguaggio formale = Problema di decisione con risposta SI/NO

1) Concetti base della teoria dei linguaggi.

Elementi della teoria della calcolabilità:

* Linguaggi ricorsivi ;
* Linguaggi ricorsivamente enumerabili ;
* Esistenza di un problema non trattabile ;
* Grammatiche = classificazione di Chomsky .

2) Linguaggi regolari:

* G di tipo 3 ;
* Automi a stati finiti ;
* Espressioni regolari ;
* Minimizzazione di automi ;
* Non determinismo .

3) Linguaggi acontestuali:

* G di tipo 2 ;
* Automi a pila ;
* Ambiguità ;
* Pumping lemma .

Concetti centrali nella teoria dei linguaggi formali

Alfabeto: insieme finito di simboli

Parola su , sequenza finita di simboli

| Esempi di alfabeti | Parole |
| --- | --- |
|  | a, aa, aaaa |
|  | 0, 1, 000, 010110 |
| (basi azotate) | DNA |
| (caso particolare) | parola vuota |

Lunghezza di una parola, numero di simboli nella parola

| Parola | Lunghezza |
| --- | --- |
|  |  |

Esempio

001011 |001011|=6

Qual’è la lunghezza di ?

= insieme delle parole su compresa

= insieme delle parole su esclusa

Concatenazione o “Prodotto di giustapposizione”

Date si dice prodotto la parola dove e

Proprietà del prodotto

* chiuso rispetto a ;
* associativo, perchè ;
* elemento neutro .

Domande

1. Lunghezza del prodotto?
2. Chi è l’elemento neutro?

Allora è un monoide

* insieme di parole
* operazione binaria associativa
* elemento neutro

Esempi

Differenza? proprietà commutativa

mentre in no

Infatti

e

La prossima lezione sarà sulla suddivisione delle parole.

Lezione dell’8 Marzo 2022

Scomposizione di parole

Esempio:

“incatenare”

Prefisso

Fattore

Suffisso

Definizione formale: dati

* Prefisso

Data , è prefisso di ?

Si dice che è prefisso di quando per una qualche parola con .

* Suffisso

Data , è suffisso di ?

Si dice che è suffisso di quando per una qualche parola con .

* Fattore

Si dice che è fattore di quando per delle date parole con .

Ci sono delle parole che possono essere contemporaneamente prefisso, suffisso e fattore di ?

* parola vuota, può essere sia prefisso, suffisso e fattore di ;
* parola stessa, può essere sia prefisso, suffisso e fattore di se stesso () .

Definizione di linguaggio formale

Un linguaggio è un qualunque sottoinsieme di :

Casi particolari

* Linguaggio vuoto ;
* Linguaggio della parola vuota .

I linguaggi si dividono in finiti e infiniti, i quali possono determinare la cardinalità.

Esempi di linguaggi finiti:

* Linguaggio vuoto, cardinalità = 0 ;
* Linguaggio con parola vuota, cardinalità = 1 ;
* ;
* Vocabolario dell’italiano .

Esempi di linguaggi infiniti:

* con ;
* Espressioni booleane .

è così definito (per induzione)

* ;
* Se allora
  + ;
  + ;
  + ;
* Nient’altro appartiene ad .

Esempi

Le parole che creiamo possiamo essere inserite all’interno di e .

Esempio

Esempi:

Parole in

Osservazione mia:

Le parole in è come se fossero le fbf del corso di logica matematica

Parole non in

Ogni volta che abbiamo una parentesi aperta bisogna avere un’altra parentesi chiusa, e dobbiamo avere solo 1 valore tra vicini tra loro.

Operazioni sui linguaggi

Insiemistiche:

* Unione
* Intersezione
* Complemento

Esercizi

1. = numeri binari escluso lo 0 = parole binarie con prefisso 1

- ;

- ;

- ;

è uguale a , ??

non appartiene né a né a , quindi

Con

Se allora

Con

Proviamo con così escludiamo “”

Per risolvere usiamo parole che contengono almeno una

Lezione del 10 Marzo 2022

Operazioni tipiche sui linguaggi:

* Prodotto

Prodotto di concatenazione preso sui linguaggi, prendendo ogni parola di e facendo il prodotto di giustapposizione con le parole di . Questo prodotto è sempre non commutativo, .

* Potenza

volte

Si prende una parola di e la si moltiplica con una parola del linguaggio , volte.

linguaggio con parola vuota

definizione ricorsiva

* Chiusura di Kleene: \*, +

Prendendo un linguaggio ,

La potenza di solito ha un indice finito

La differenza tra le due chiusure è la parola vuota.

e derivano proprio dalla chiusura di Kleene.

Esercizio

(In teoria no, dovrebbe essere meno il linguaggio con la parola vuota.

Esercizi

1)

il prodotto di due linguaggi non è commutativo perchè neanche il prodotto di due parole lo è (giustapposizione).

2) Numeri binari:

e

Il linguaggio contiene solo il simbolo mentre il linguaggio contiene tutte le parole del linguaggio.

(la seconda formula è un abuso di notazione)

Così otteniamo i numeri binari.

3)

Concatenazione delle parole del linguaggio data una potenza , ad esempio data , appartengono a , mentre ad esempio non appartengono a .

Inizialmente lo abbiamo considerato come dogma.

Consideriamo la precedente chiusura di Kleene,

parole su con lunghezza , lo abbiamo già fatto con il nostro esempio , il quale rappresentava un alfabeto. Quindi le chiusure di Kleene funzionano anche sui alfabeti.

Quindi l’uguaglianza è vera!!

Esercizi



( , linguaggio formato dalle parole composte dalle parole del linguaggio di , in qualsiasi ordine con una lunghezza finita)

e

Osservazione:

Consideriamo una parola appartenente a , ad esempio , la domanda da farci è: “In quanti modi posso scomporre la parola con le parole del linguaggio ?”

Infatti è uguale a “” , “” , “”

Decomposizione in

Se una parola in ammette più scomposizioni per la stessa parola, allora il linguaggio non sarà un codice.

Concetto di codice

Possiamo costruire dei codici come dei linguaggi formali, soprattutto utilizzando il + della chiusura di Kleene (quindi ). Per avere un codice ho bisogno di una proprietà fondamentale, cioè se dato un linguaggio , una qualsiasi parola in possiede un, ed un solo, modo per essere scomposta.

Un codice è, e deve essere un linguaggio finito.

Definizione

Un linguaggio è un codice quando:

Ogni parola in è decomponibile in un unico modo in parole di .

Esempio:

E’ un codice

Prendiamo ad esempio

Questa parola può essere decomposta in un unico modo attraverso il nostro linguaggio, quindi è un codice.

Definizione

è un codice Prefisso o Istantaneo quando:

* è un codice ;
* ogni parola di non è prefisso di altre parole di .

Proprietà dei codici prefissi:

Ammettono un algoritmo di decodifica on-line (mentre sto leggendo la parola stessa posso decodificare man mano che procedo), quindi è istantaneo.

Esempi

è un codice prefisso

non è un codice prefisso, dato che la parola “0” è prefisso della parola “01”

Codice ASCII esteso

Codifica ogni carattere della tastiera

In sequenze di 8 bit:

Chi è ?

Perchè è importante che sia un codice?

Risposta:

File a caratteri file binario file a caratteri originario (non accadrebbe se non fosse un codice, quindi non otterremmo lo stesso file originale durante la decodifica)

è prefisso (o istantaneo):

* Perchè la decodifica non deve impiegare troppo tempo ;
* Perchè le parole di hanno tutte la stessa lunghezza, quindi è impossibile che una di quelle parole sia prefisso di un'altra (una parola è prefisso solo di se stessa).

L’algoritmo di decodifica on-line avviene attraverso il taglio del file ogni 8 bit.

Lezione del 15 Marzo 2022

Riprendiamo la definizione di codice.

Un linguaggio è un codice se ogni parola in è un codice decomponibile in un unico modo in parole di .

Esempio negativo

Può essere scomposta come:

* ;
* ;
* ;
* .

Quindi non è un codice

Esempio positivo

è un codice

Si dice che un codice prefisso o istantaneo se ogni parola di non è prefisso delle altre parole di .

Esempio negativo

Codice non prefisso siccome è prefisso di

Esempio positivo

ogni sua parola non è prefisso di nessun’altra

Codice ASCII esteso

Codifica i caratteri della tastiera in sequenze di 8 bit.

Perchè è importante che sia un codice?

Risposta

Stiamo lavorando sul nostro file a caratteri, salvando il file esso viene codificato in una serie di caratteri in codice ASCII in un file binario. Ad un certo punto riapro per apportare altre modifiche, avviene la decodifica di questo file. Se non fosse un codice otterremmo un file diverso da quello originario.

File a caratteri file binario file a caratteri originario

Ma visto che è un codice otteniamo il file originario.

* è prefisso perchè ogni parola ha lunghezza di 8 bit e hanno almeno 1 bit di differenza tra loro.
* Dato che è prefisso ammette un algoritmo di decodifica on-line (linea per linea), il quale taglia il file binario ogni 8 bit e converte carattere per carattere.

Un carattere può essere visto come un problema

Definizione di linguaggi

1. (definizione possibile solo con linguaggi finiti) definizione estensiva ;
2. Dove la proprietà viene soddisfatta solo se , definizione intensiva (si può usare anche con i linguaggi infiniti) .

Fatto:

Ad ogni è associato il problema

Linguaggio Problema con Input: e Output: soddisfa la proprietà ? SI / NO (Problema di decisione)

Dato siamo interessati a:

1. Sapere se ammette una soluzione automatica ;
2. Se ammette un algoritmo, se si trova quello migliore .

Tutto questo si traduce nella teoria dei linguaggi in:

1. Sapere se ammette un sistema formale, che può essere un

| Sistema generativo | Sistema riconoscitivo |
| --- | --- |
| Genera le parole di | Stabilisco se una parola appartiene a oppure no |

1. Se ammette un sistema riconoscitivo, trovare quello migliore .

Esempi

Consideriamo

Software in rete si chiedono se un indirizzo è corretto, cioè se Trovare un sistema riconoscitivo per .

Altro esempio

Linguaggio

Si richiede che queste password vengano generate in maniera automatica, o sistematica Trovare un sistema generativo per

Domanda: tutti i problemi di decisione ammettono una soluzione automatica?

La risposta è stata data nella teoria di calcolabilità (NO, risultato indipendente dalla tecnologia).

Per mostrare il risultato abbiamo bisogno di alcuni concetti base:

* Procedura = sequenza finita di istruzioni che possono portare a un risultato ;
* Algoritmo = procedura che termina su ogni input .

Stabiliamo il concetto di programma:

1. Aspetto sintattico, un programma è una parola binaria grazie al codice ASCII ;
2. 2) Aspetto semantico, un programma associa a degli input, degli output. Quindi è una legge che allo stesso input associa sempre lo stesso output. Quindi essa è definibile come una funzione. programma con risultato di su input .

Osservazione

Un qualsiasi input per il programma è binario : , ma anche Quindi anche può essere visto come un dato e essere passato ad un altro programma in input.

Limitazione sui programmi

* O i programmi non terminano ;
* O se terminano, danno in uscita o 1 o 0 (Uscita a 1 bit, problemi di decisione) .

Notazione

su input non termina

su input termina quindi (Output uguale a 0 o uguale a 1)

Esempio

Problema: calcolare la parità dei numeri binari positivi.

Quindi la parità di stringhe della forma

Programma

(cifre del numero binario )

|  | { | 1 se è binario pari |
| --- | --- | --- |
| 0 se è binario dispari |
| se non è binario |

|  | { | 1 se |
| --- | --- | --- |
| 0 se |
| se |

Associazione Input/Output

Lezione del 17 Marzo 2022

Definizione

La funzione caratteristica di è:

|  | { | 1 | se |
| --- | --- | --- | --- |
| 0 | se |

Esiste sempre per ogni !

Definizione

Un linguaggio è detto ricorsivo quando esiste un algoritmo tale che:

|  | { | 1 | se |
| --- | --- | --- | --- |
| 0 | se |

La differenza consiste nel fatto che la prima è una funzione, mentre la seconda è un algoritmo che supporta una funzione simile alla funzione caratteristica. Infatti la seconda tira in ballo una funzione, quindi un algoritmo. Nel primo caso la funzione c’è per ogni linguaggio, mentre per il secondo abbiamo bisogno dell’esistenza di un algoritmo di risoluzione. L’algoritmo non fa altro che calcolare la funzione caratteristica, la quale nel caso dei linguaggi ricorsivi verrà descritta come un algoritmo.

Inoltre

Se è ricorsivo allora:

* è detto decidibile (possiamo associare la parola “ricorsivo”, proprietà che appartiene al linguaggio, con “decidibile”, proprietà che appartiene al problema del linguaggio) ;
* ammette un sistema riconoscitivo (parliamo di un sistema formale per riconoscere gli elementi) .

Esempi

Problemi decidibili:

* numeri pari ;
* numeri primi ;

Linguaggi ricorsivi:

* ;
* .

Definizione

Un linguaggio è ricorsivamente enumerabile quando esiste una procedura tale che:

|  | { | 1 | se |
| --- | --- | --- | --- |
|  | se |

Il programma può anche non terminare.

(Non si può dire che non appartiene).

Se è ricorsivamente enumerabile allora:

* è detto semidecidibile ;
* ammette un sistema generativo .

Relazione:

L’insieme dei linguaggi ricorsivi è un sottoinsieme dei linguaggi ricorsivamente enumerabili. Dire che è ricorsivamente enumerabile significa che l’algoritmo ha anche uno stato (o meglio “ramo”) di loop.

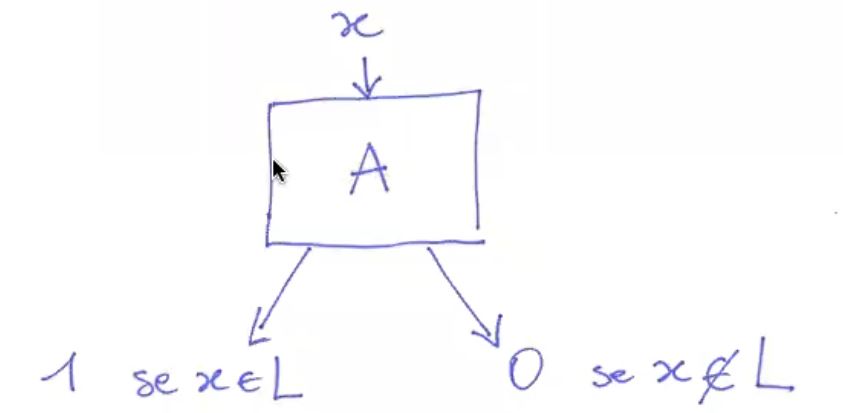


Teorema

Se è ricorsivo è ricorsivamente enumerabile

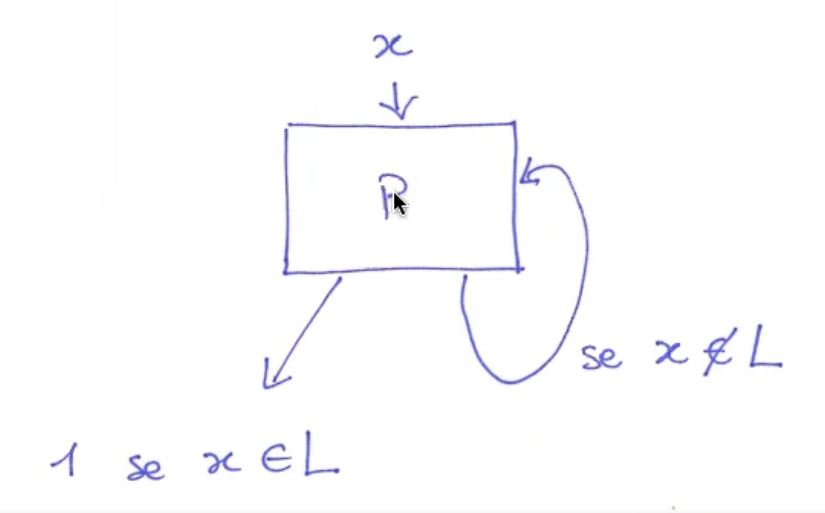
Dimostrazione

Per ipotesi esiste un algoritmo per :



devo dimostrare l’esistenza di una procedura :

(prendendo l’algoritmo precedente e peggiorarlo nel ramo 0)



Procedura :

Dimostrazione di correttezza

segue che è ricorsivamente enumerabile

Domanda: è vero il viceversa? NO, intuitivamente non sempre si può trasformare una procedura in un algoritmo.

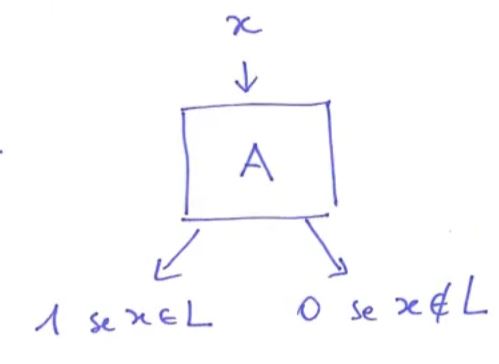
Teorema

Se è ricorsivo è ricorsivo

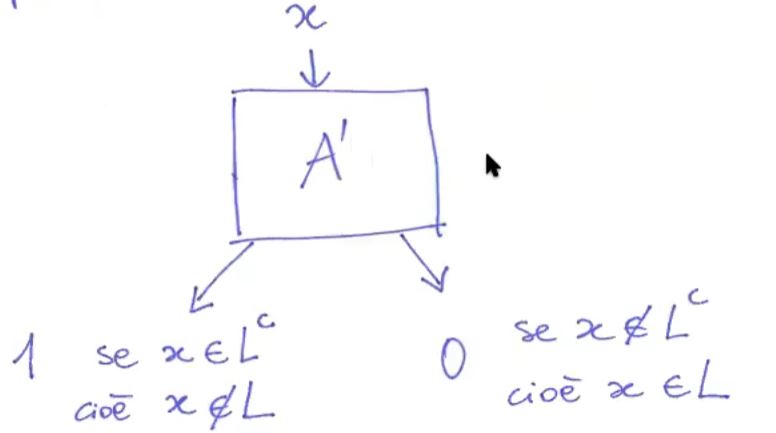
(se so riconoscere un linguaggio, so riconoscere anche cosa non è di quel linguaggio)

Dimostrazione

Per ipotesi ho:

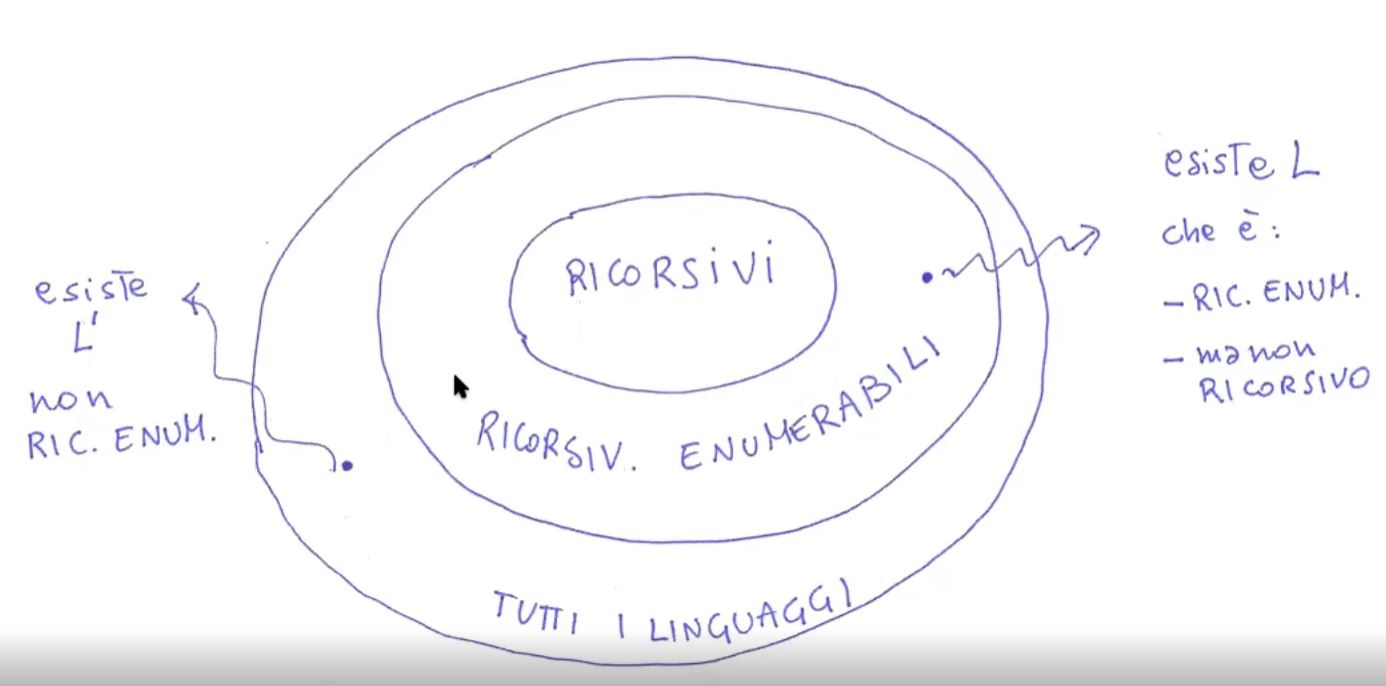


devo quindi costruire un algoritmo per



(in pratica prendiamo e invertiamo gli output)

Risultati importanti della teoria della calcolabilità



Conseguenze di questi risultati:

* Non è possibile verificare per via automatica la correttezza dei programmi (Teorema di Rice, 1950) oppure non è possibile verificare per via automatica l’equivalenza di due programmi cioè: dati e verificare che ;
* Non è possibile verificare per via automatica la terminazione dei programmi cioè:
  + Problema dell’arresto:

Input: programma , dato

Output:

questo problema è indecidibile! (Turing, 1936)

* Ci sono “Teoremi” matematici non dimostrabili (Risultato di incompletezza di Goedel, 1930) .

Interprete, programma

In input passiamo ad la coppia (programma, dato) e in uscita ho il risultato dell’esecuzione del “programma” sul “dato”

|  | { |  | se è un programma |
| --- | --- | --- | --- |
|  | altrimenti |

Lezione del 22 Marzo 2022

Linguaggi Ricorsivi o Ricorsivamente Enumerabili

* Un linguaggio si dice Ricorsivo () quando esiste un algoritmo tale che:

|  | { | 1 | se |
| --- | --- | --- | --- |
| 0 | se |

Problemi decidibili, grazie ai quali riesco a discriminare le parole che appartengono al linguaggio da quelle che non gli appartengono.

* Un linguaggio si dice Ricorsivamente Enumerabile () quando esiste una procedura implementata da un programma tale che:

|  | { | 1 | se |
| --- | --- | --- | --- |
|  | se |

Queste sono 2 classi di linguaggi.



Ricorsivo Ricorsivamente Enumerabile

Posso dimostrarlo

Per il linguaggio Ricorsivo posso prendere un algoritmo e trasformarlo in una procedura (che va in stato di loop invece di restituire 0).

è Ricorsivamente Enumerabile ma non Ricorsivo (devo dimostrarlo esibendo un linguaggio).

Linguaggio dell’arresto, che dato in input un programma e un dato ci dice se esso termina o no, in questo caso esso è semidecidibile ma non decidibile.

Interprete: è un programma “”

Input: passiamo la coppia (programma, dato)

Output: il risultato del programma su quel determinato dato

|  | { | quando è un programma |
| --- | --- | --- |
| altrimenti |

Dove è l’interprete, è il programma, è il dato e

Definiamo il linguaggio

Insieme delle parole su che passate sia come dato che come programma all’interprete fanno in modo che esso termini.

è quindi lecito passarlo ad interprete.

è detto linguaggio dell’arresto ristretto (perchè testa solo ).

Teorema

1. è un linguaggio Ricorsivamente Enumerabile ;
2. non è ricorsivo ;
3. non è ricorsivamente enumerabile.

Dimostrazione del punto 1

Devo esibire per una procedura tale che

|  | { | se |
| --- | --- | --- |
| se |

quando

quando

Posso costruire la seguente procedura

\\ istruzione eseguibile, dato che esiste sempre e quindi posso richiamarlo

(Non termina se non termina)

è proprio la procedura che cercavamo.

Dimostrazione di correttezza di :

* viene eseguita l’istruzione GIUSTO!!
* va in loop all’istruzione GIUSTO!!

è la procedura che cercavamo che mostra che è un linguaggio Ricorsivamente Enumerabile.

Dimostrazione del punto 2

non è ricorsivo

Tecnica dell’assurdo

Supponendo che sia Ricorsivo, quindi posso costruire un programma che mi porterà ad una contraddizione.

Procedura

Programma fattibile

Posso testare perché sto supponendo che sia ricorsivo. Tutte le istruzioni sono implementabili. Esiste codice binario per la codifica di con

Allora passo ad il codice binario di .

Quanto vale ?

Secondo alcune considerazioni ottengo la seguente risposta:

per definizione di è vero.

per definizione di e di

Ottenendo così

Esiste un altro modo per mostrare quanto vale , considerando il valore stesso del codice:

* Caso per definizione di (uguale a o a ) , quindi ma per la proprietà precedente ottenendo così o , quindi ottenendo un ASSURDO!! ;
* Caso per definizione di , quindi , che diventa poi ottenendo un ASSURDO!!

Quindi non esiste e non è il codice di un programma.

Dato che non esiste, quindi non posso fare , il programma non termina (non avendo un linguaggio ricorsivo) e quindi non è ricorsivo.

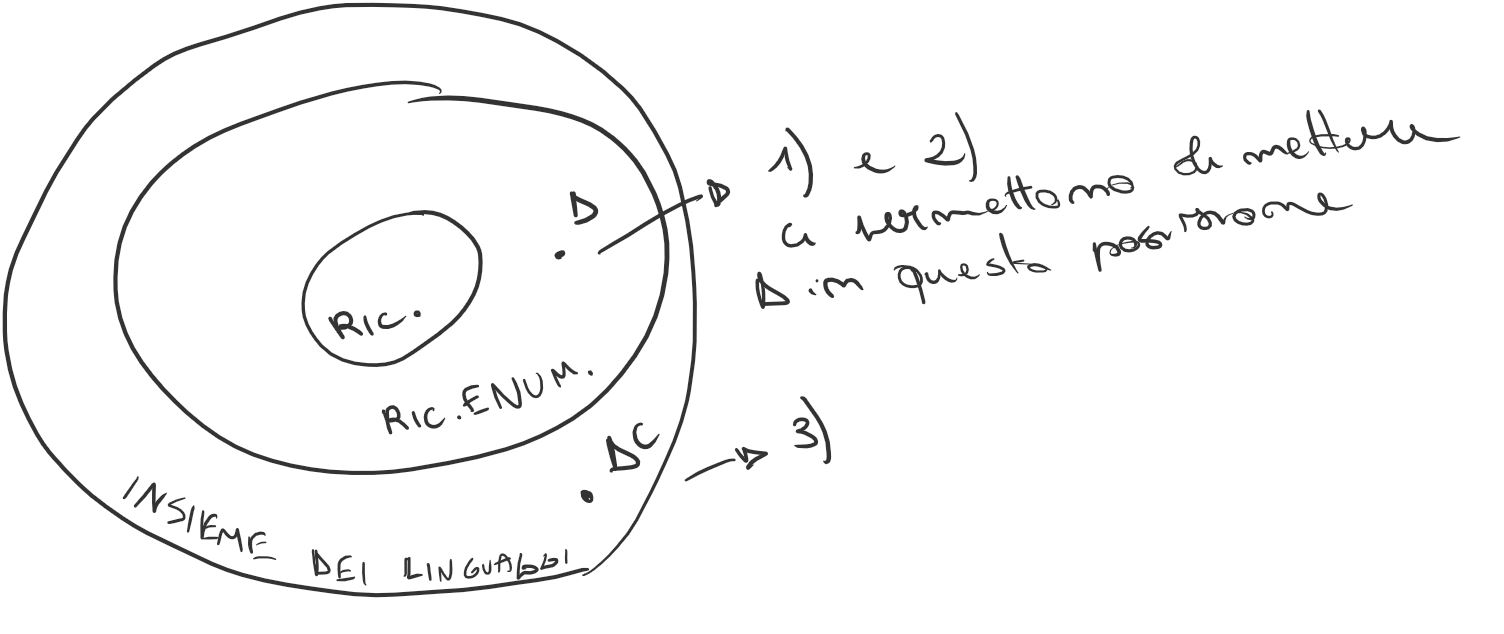
Lezione del 24 Marzo 2022

Linguaggio dell’arresto ristretto

In ci sono sia i “non programmi”, sia i programmi che non terminano su loro stessi.

Parte 3 del teorema

* non è ricorsivamente enumerabile.

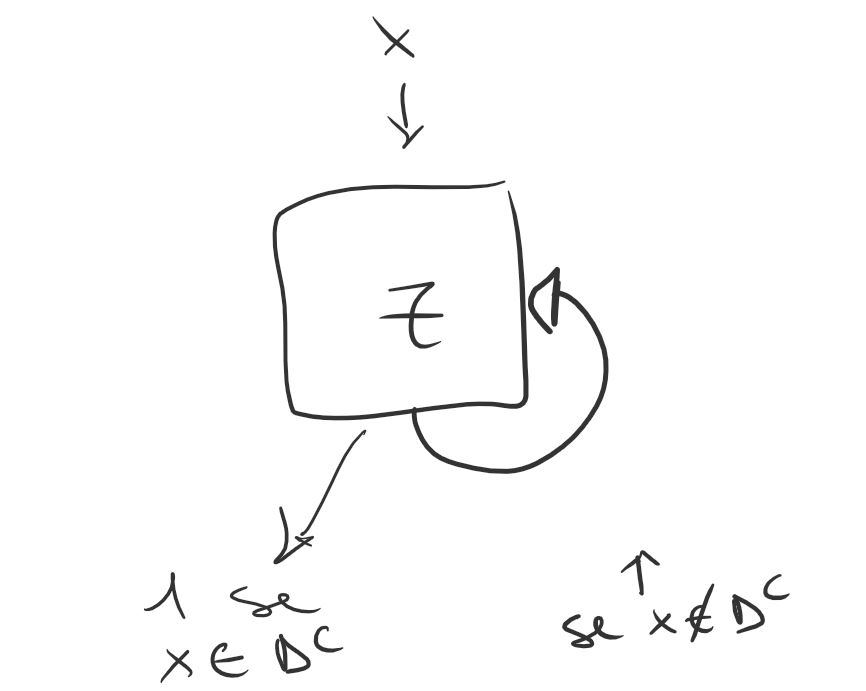


Adesso dimostriamo il punto 3 del teorema

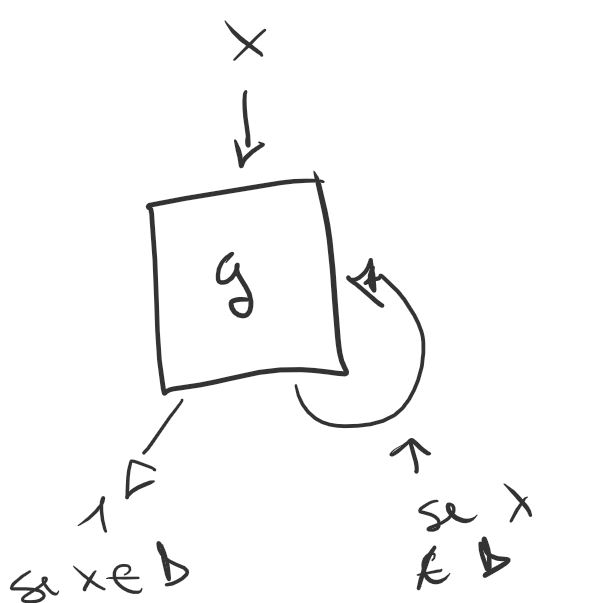
Non è Ricorsivamente Enumerabile

Dimostrazione per assurdo

Suppongo che sia Ricorsivamente enumerabile esiste una procedura tale che:

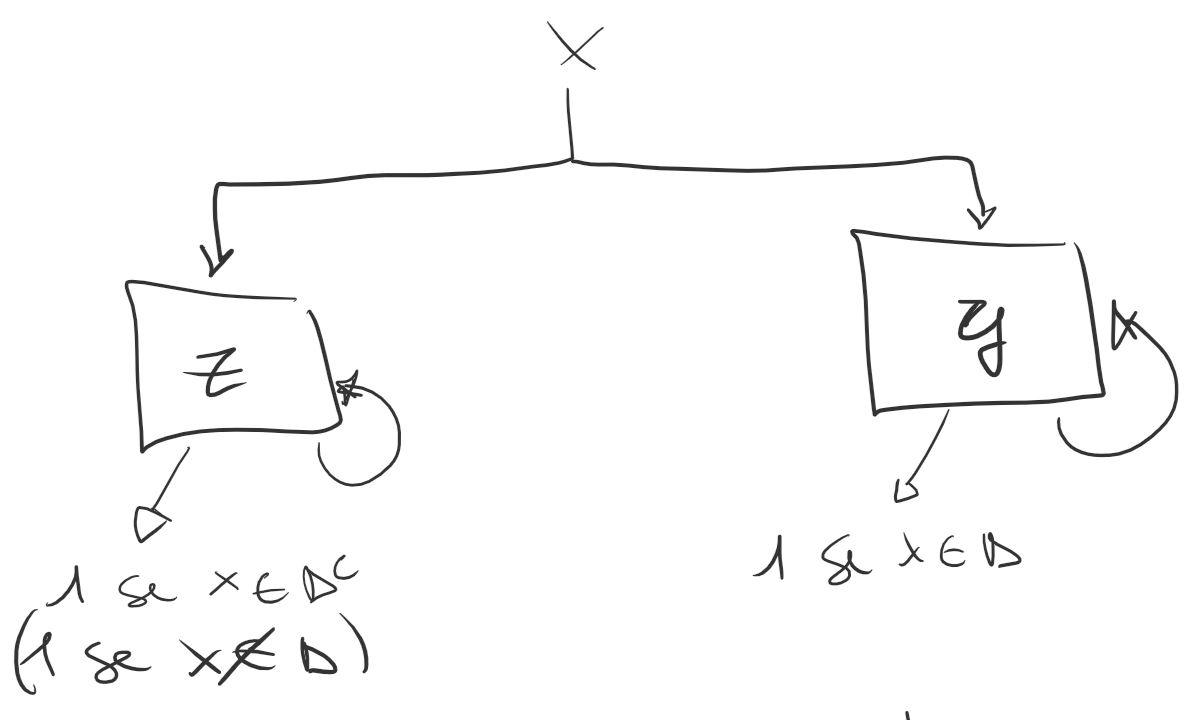


Inoltre anche è ricorsivamente enumerabile (per il punto 1 del teorema) e pertanto esiste una procedura tale che:



Idea

Adesso prendo



Sembra un algoritmo Ricorsivo che farebbe diventare a sua volta ricorsivo.

Programma

\\ numero di passi

// quando esco dal while significa che una delle due ha terminato, ma non sappiamo quale, dato che restituiscono entrambe

Domanda: cosa fa ?

* Se sono nel caso in cui termina e esiste un ;
* Se sono nel caso in cui termina e esiste un .

è un algoritmo per è ricorsivo ASSURDO!!

Contraddizione perchè il codice non può esistere, dobbiamo trovare l’errore ma tutte le procedure e funzioni sono vere tranne la nostra supposizione non è Ricorsivamente Enumerabile.

Abbiamo dimostrato tutti i punti del Teorema.

Domanda: esiste un problema che non ammette soluzione algoritmica ?

Esiste un problema indecidibile ?

Risposta: SI!

Problema dell’arresto o della fermata

Input:

Output: Risposta alla domanda ?

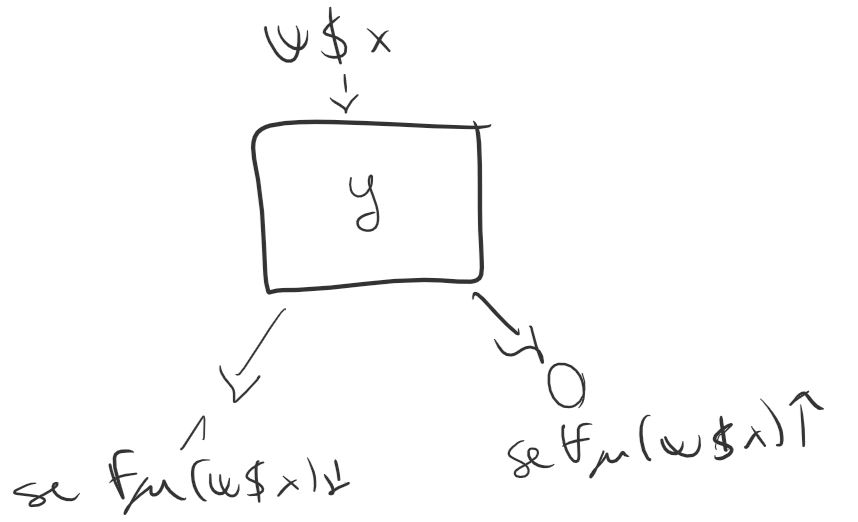
Linguaggio dell’arresto

Teorema

non è ricorsivo

Dimostrazione per assurdo

Suppongo Ricorsivo e pertanto esiste :



Allora posso costruire un programma che chiameremo

se quindi se e se quindi se

Programma

Cosa fa ?

Se ;

Se .

Il test però è fattibile solo nel momento in cui sia ricorsivo. è un algoritmo per assurdo non è ricorsivo anche non è ricorsivo.

Il problema dell’arresto è indecidibile e indipendente dalla tecnologia.

Lezione del 29 Marzo 2022

In questa lezione iniziamo a occuparci dei sistemi formali per l’automatizzazione per risolvere i problemi di decisioni. Lo facciamo attraverso i linguaggi formali. Il primo sistema che può essere automatizzato per ottenere risultati nei problemi di decisione sono i sistemi generativi.

Un particolare sistema generativo:

La grammatica

Si parla delle regole sintattiche (o di produzione) attraverso le quali ottengo le parole del linguaggio.

Consideriamo un sottolinguaggio dell’italiano: gli annunci ferroviari in stazione

Se abbiamo stiamo considerando delle regole

La soppressione mi permette di generare la frase in stazione in maniera automatica, la quale verrà poi annunciata dagli altoparlanti.

Numero di passanti dei treni di Milano.

Le ore e i minuti corrispondono a quelle reali correnti.

viene chiamato “variabile” o “metasimbolo” (deve essere sostituito)

viene chiamato “terminale” (ed è fisso, non è sempre vero)

è il simbolo generale detto “assioma” (porta il nome della frase che vogliamo generale)

è una regola detta di “produzione”

indica l’applicazione di una regola ed è chiamato “passo di derivazione” (quando applichiamo una regola della grammatica, nel nostro esempio non sono stati ancora applicate le regole della grammatica)

è l’applicazione di più regole detto “derivazione in zero o più passi” (più passi di derivazione insieme

Esempio:

Questa è una derivazione per la generazione di una frase degli annunci ferroviari.

Formalizziamo

Definizione:

Una grammatica è una quadrupla:

dove

* insieme finito dei simboli terminali ;
* insieme dei metasimboli con ;
* è un simbolo in , ed è l’Assioma (start symbol) ;
* è l’insieme finito delle regole di produzione .

Definizione di regola di produzione

dove : e

unione di due alfabeti, ottenendo un nuovo alfabeto

Esempio

ma non

Definizione di passo di derivazione

Si dice che è derivabile da in un passo e si scrive

(Applicazione di una regola appartenente a )

Quando:

e e

Allora

Definizione di derivazione in zero o più passi

Si dice che è derivabile da in zero o più passi e si scrive

Quando:

esiste e esistono le parole tali che

Oppure

Definizione

Il linguaggio generato da è:

(Tutte le parole generate dall’assioma principale)

Osservazioni:

* Un linguaggio può ammettere più grammatiche che lo generano ;
* Due grammatiche e si dicono equivalenti se .

Esempi

Esempio 1.

Esempio 2.

Se

Qui possiamo derivare due grammatiche

e sono equivalenti

Esempio 3.

Esempio 4.

Palindrome:

Regole:

Lezione del 31 Marzo 2022

Continuiamo con gli esempi della volta precedente

Esempio 5.

numero di simboli in

:

: mentre quindi non appartiene al linguaggio

(con questo spostiamo l’ordine dopo aver creato una parola simmetrica)

Esempio 6.

Queste sono espressioni libere da contesto, sono più difficili da utilizzare perchè per trasformare e bisognerebbe utilizzare espressioni dipendenti dal contesto (con simboli terminali all’inizio) quindi useremo:

Domanda:

Tutti i linguaggi formali ammettono una grammatica?

Non tutti, infatti abbiamo il seguente teorema

Teorema

è ricorsivamente enumerabile è generato da

Dimostrazione (solo )

Data la grammatica per posso costruire una procedura di tale che

|  | { | se |
| --- | --- | --- |
| se |

Definizioni:

Nota :

(Quelle con variabili e simboli terminali sono contenute in mentre quelle con solo simboli terminali )

Fatto:

Infatti:

1. è il numero di passi di derivazione in ();
2. in “” di derivazione () .

Formalizzo e :

(formalizzazione ricorsiva)

Inizialmente costruisco la procedura che genera tutte le parole di

stampa:

. . .

Procedura

// fisso

Funzioni di :

Adesso trasformiamo nella procedura :

modifica :

Passo in input ad

modifica :

Sostituisco con

Correttezza della procedura modificata:

verrà eseguito

il ciclo andrà in loop

Linguaggio delle espressioni booleane su e

Definizione

* ;
* ;
* nient’altro appartiene ad .

Grammatica di :

Lezione del 5 Aprile 2022

Introduciamo il linguaggio delle espressioni booleane e la sua grammatica in modo da costruire e dare una definizione dell’albero di derivazione.

Linguaggio delle espressioni booleane su e su

Definizione

* ;
* e ;
* Nient’altro appartiene ad .

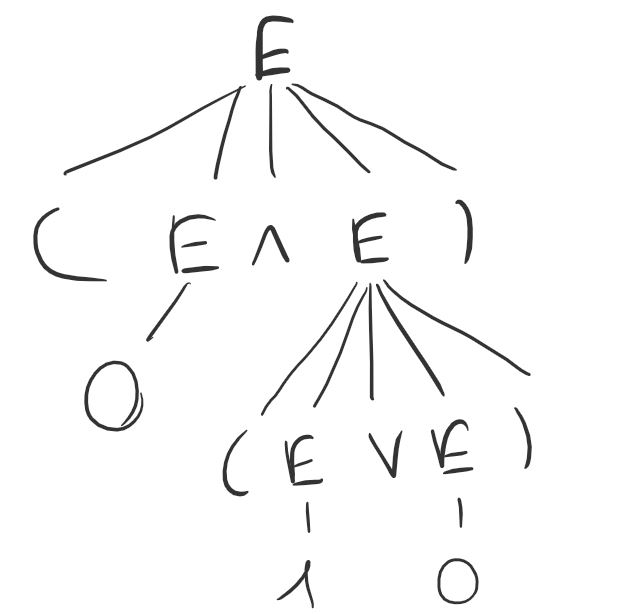
Grammatica del linguaggio

* ;
* variabile ;
* assioma ;
* .

Prendiamo adesso ad esempio la parola

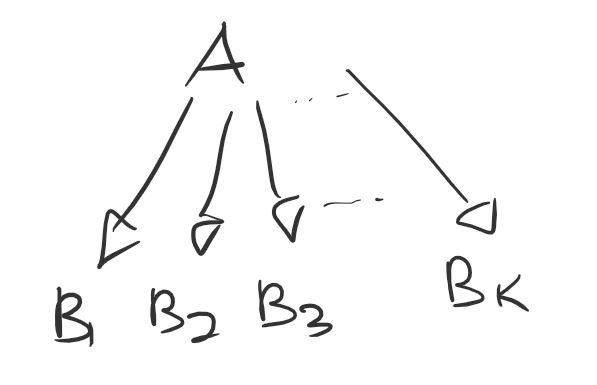
Si parte con l’operazione più esterna.

Posso rappresentare la derivazione di attraverso il suo albero di derivazione:



Esso è un oggetto ben preciso con alla radice l’assioma della grammatica, i nodi sono le variabili della grammatica e le foglie sono i simboli terminali tali che se letti da sinistra verso destra ottengo .

I sottoalberi sono quelli delle regole di derivazione.

 di

Osservazione

Per avere alberi di derivazione, deve essere almeno di tipo (non ancora introdotto), vuol dire che le regole in testa devono avere una sola variabile.

Classificazione delle grammatiche

(versione preliminare)

I tipi si differenziano per le regole. Ci sono tipi in funzione di esse:

* Tipo 0. Non abbiamo nessun vincolo sulle regole di produzione ;
* Tipo 1. Sono ammesse solo le regole con (quindi non possono produrre ) ;
* Tipo 2. Sono ammesse regole della forma dove e ;
* Tipo 3. Sono ammesse regole della forma e dove e sono delle variabili, e .

Si va dalla più complessa alla più semplice (meno regole più complicata).

Fatto: Una grammatica di tipo lo è anche di tipo (per ed è ricorsivo come fatto).

Definizione

Un linguaggio si dice di tipo se ammette una grammatica di tipo ().

Definizione

(classi di linguaggi di tipo , ce ne sono )

linguaggi regolari

linguaggi liberi da contesto

linguaggi definiti da contesto

linguaggi ricorsivamente enumerabili

Vediamo esempi di grammatiche

(del tipo mentre per il tip va bene qualsiasi grammatica)

Tipo 3: con e ,

* per ;
* per .

Tipo 2: con e

* per ; (Osservazione: del tipo 3) ;

Tipo 1: con

* Per il linguaggio con .

Esempio concreto di di tipo , i DTD (grammatica del tipo ) dei documenti XML (linguaggio di tipo )

Documenti XML

Documenti composti da “testo” e “tag”. Il testo può essere arbitrario mentre il tag deve rispettare alcune regole.

Tag= marcatore di testo che consente di dargli informazioni semantiche.

Esempio

indicano inizio e fine della rubrica, per ogni tag aperto ci deve essere un tag chiuso corrispondente

Formalizziamo le regole

Tag aperto:

Tag chiuso:

Condizioni che un documento deve soddisfare sono:

1. Deve esistere un unico tag che contiene tutto il documento nell’esempio avevamo ;
2. Ogni tag aperto deve essere seguito dal corrispondente chiuso ;
3. I tag devono essere innestati correttamente ().

Definizione

Un documento per il quale valgono le condizioni si dice “corretto”.

DTD = definisce come i tag possono essere innestati tra di loro

Esempio:

Nel documento “rubrica” il tag non può stare dentro al tag .

Definizione

Un documento XML è valido secondo un certo DTD se è generato da quel DTD (grammatica di tipo ). Le regole di un DTD sono della forma:

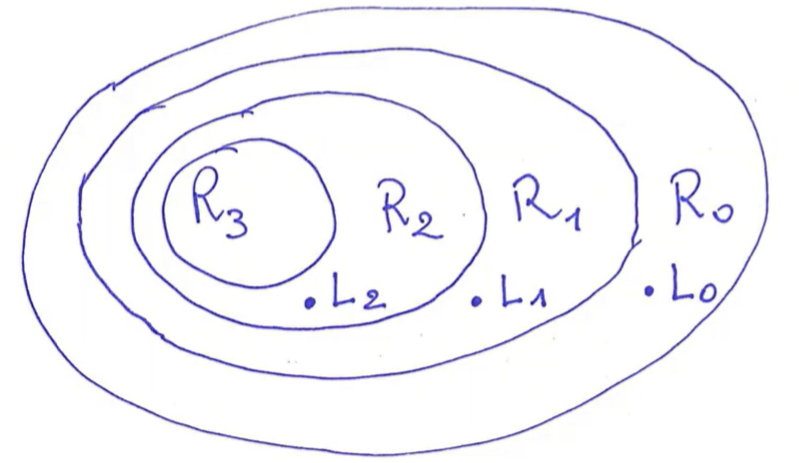
dove espressione di variabili con le operazioni

Creiamo un DTD:

Anche questo DTD è della forma

Lezione del 7 Aprile 2022

Teorema sugli :



“sottoinsieme di” o “inclusione propria” (si esclude la possibilità che le classi siano uguali tra di loro, ci sarà un linguaggio appartenente a ma non a con )

classe di linguaggio con tipo di grammatica associato

linguaggi regolari

liberi da contesto

dipendenti da contesto

linguaggi ricorsivamente enumerabili

Dimostrazione

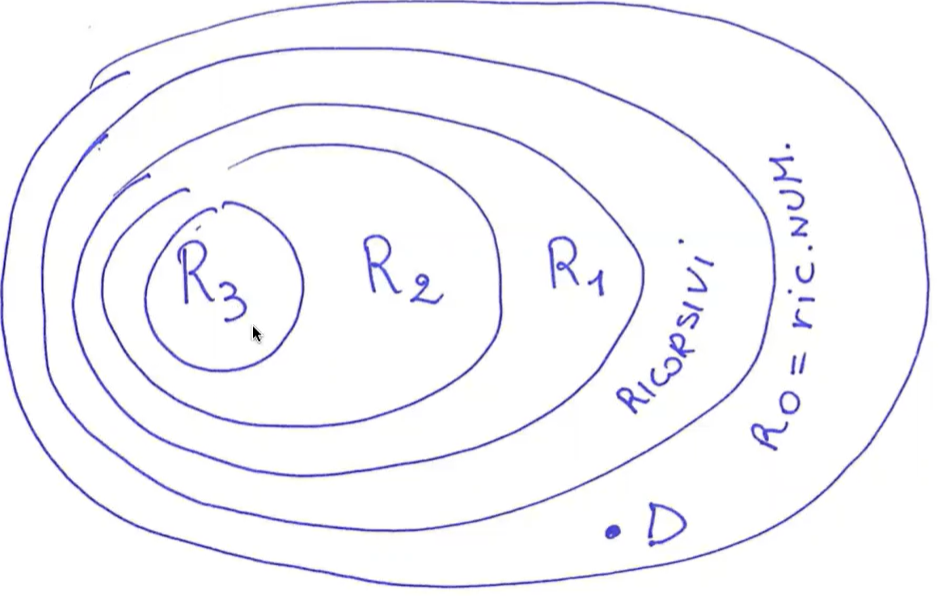
Separiamo la dimostrazione dell’inclusione da quella dell’esistenza di e .

* L’inclusione tra gli segue dal FATTO che abbiamo dato sui tipi di : ;
* dimostriamo che l’inclusione è propria ;
  + esiste ma ad esempio . Infatti esso ammette di tipo 2 e inoltre perchè non ammette un automa a stati finiti che lo riconosce ;
  + esiste ma ad esempio . Infatti esso ammette di tipo 1 e perchè non soddisfa il pumping lemma per i linguaggi liberi da contesto ;
  + esiste ma ad esempio che ammette di tipo 0 ma .

Dimostrazione

Passo 1.

Passo 2. non è ricorsivo e è ricorsivamente enumerabile



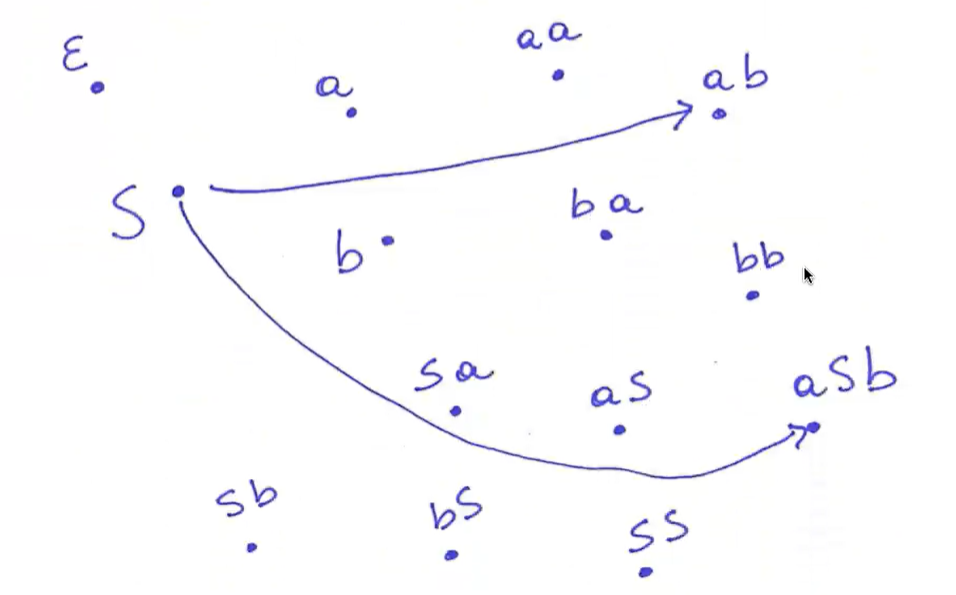
Abbiamo dimostrato (ovvio) il passo 2 ma non il passo 1

Dobbiamo definire un programma creato con grammatica di tipo 1 per mostrare che sia ricorsivo.

Definizione di con programma per i grafi.

dove:

(in generale gli archi sono coppie di vertici)



Algoritmo

// serve

Osservazioni:

Il problema della generazione di si trasforma nella ricerca di un cammino in un grafo. Il tempo dell’algoritmo è esponenziale, richiesto da .

Correttezza di :

esiste una derivazione in

tale che

per ogni abbiamo che , e inoltre abbiamo che

Lezione del 12 Aprile 2022

In questa lezione vedremo la classificazione di Chomsky definitiva.

Il motivo per cui quella precedente va bene è per la parola vuota (che per adesso è generata solo dalle grammatiche di tipo 0).

Data la classificazione di Chomsky provvisoria:

Se è di tipo , di che tipo è ?

Avrei bisogno di (regola che permette di generarla) consentita solo dal tipo 0.

Quindi è del tipo 0.

Se abbiamo un linguaggio di altro tipo e aggiungo cambia? Se cambia, questa classificazione non è ragionevole perchè una singola parola non può cambiare il tipo del linguaggio. Pertanto apportiamo 2 modifiche.

modifica:

Riguardo le grammatiche di tipo applichiamo la seguente modifica.

E’ ammessa la regola a patto che:

* sia l’assioma ;
* non compaia sulla destra di altre regole.

FATTO :

Se è di tipo , allora anche è di tipo .

Dimostrazione

Ho grammatica per di tipo e devo costruire per mantenendo il tipo.

Introduco (nuovo assioma) in e le regole e .

non è presente alla destra di altre regole.

La grammatica rispetta la nostra modifica ed è dello stesso tipo di .

modifica:

Per le grammatiche di tipo ammetto regole della forma dove è una variabile arbitraria.

Cancellando la variabile posso accorciare la parola in corso di derivazione (permettendo una derivazione decrescente).

FATTO :

Se ho di tipo con regole arbitrarie, posso ottenere equivalente di tipo che rispetta .

Esempio

Questa grammatica è di tipo e non rispetta (perchè compare alla destra della prima e della seconda regola).

Applico alle altre regole:

Aggiungo queste regole e elimino (perchè le regole appena inserite simulano ).

Manca la possibilità di iniziare generando . Per generare la parola vuota ho di nuovo bisogno di e di .

Quindi :

Il tipo è preservato e in più la nuova grammatica rispetta .

Classificazione di Chomsky definitiva

* Tipo 0: regole arbitrarie ;
* Tipo 1: regole della forma

con con l’eccezione se :

* è l’assioma ;
* non compare sulla destra di altre regole.
* Tipo 2: regole della forma con e invece di includendo quindi anche ;
* Tipo 3: regole nella forma con e (quindi adesso è compresa anche per ) .

Teorema sugli

continua a valere grazie al FATTO

Teorema

Continua a valere perchè la frase preserva la lunghezza non decrescente delle derivazioni.

Forme equivalenti per il tipo 3:

dove variabili e caratteri terminali ;

dove variabili e ;

dove variabili e solo se .

Posso passare dalla forma a eliminando le regole della forma .

* Introduco una variabile ;
* Per ogni regola introduco e ;
* Cancello le regole del tipo .

Posso passare da a eliminando le regole della forma .

* Per ogni regola della forma aggiungo regole della forma se ;
* Cancello le regole della forma .

E’ possibile mostrare la seguente equivalenza:

| (forma lineare a destra) |  | (caso particolare della lineare a destra) |
| --- | --- | --- |

) Facile perchè

) Trasformazione da mostrare

Esempio

con simboli terminali

Prendo

Prendo

Abbiamo trovato regole con rendendole

Per le regole della forma con utilizziamo la stessa tecnica

Ad esempio

con lettere terminali

Prendo

Prendo

Caso non trattato

Le regole della forma con cioè .

Posso sempre eliminarle aggiungendo regole della forma e (La prova non la facciamo)

Lezione del 21 Aprile 2022

Teoria degli automi a stati finiti

Applicazioni: HW e SW

* HW: progettazione di circuiti elettronici modellando semplici sistemi ;
* SW: costruzione di analizzatori lessicali che fanno da soluzione a problema di ricerca nei testi o modellare pagine web.

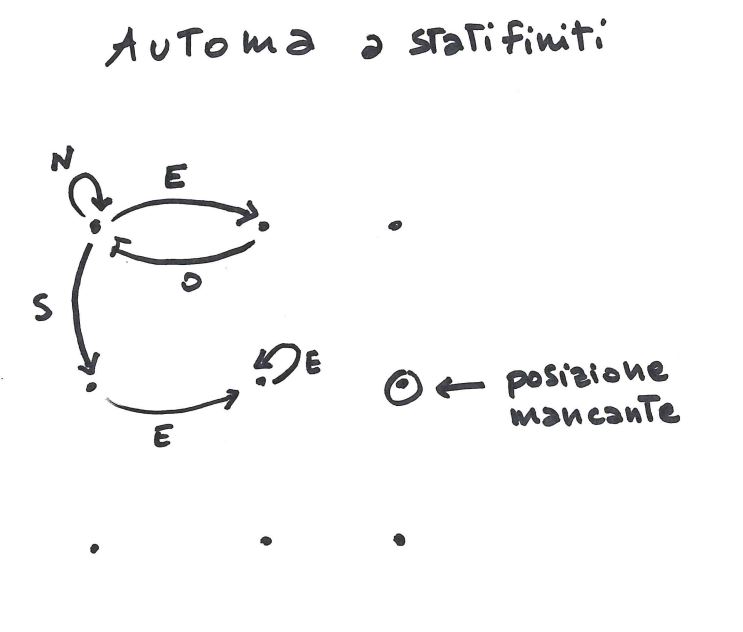
Un sistema da modellare con automi:

* Abbiamo un Robot che si muove su una griglia ;
* Il Robot è telecomandato da segnali che indicano la direzione ;
* Il comportamento del Robot può essere modellato da un automa.

Postazioni nella griglia stati dell’automa.

Segnali inviati al Robot simboli di input dell’automa: (direzionali)

Questo ci porta al seguente automa:



Da cosa dipende la posizione del Robot?

Caratteristica principale:

La posizione prossima del Robot dipende:

* dalla posizione attuale ;
* dal simbolo inviato.

Osservazione:

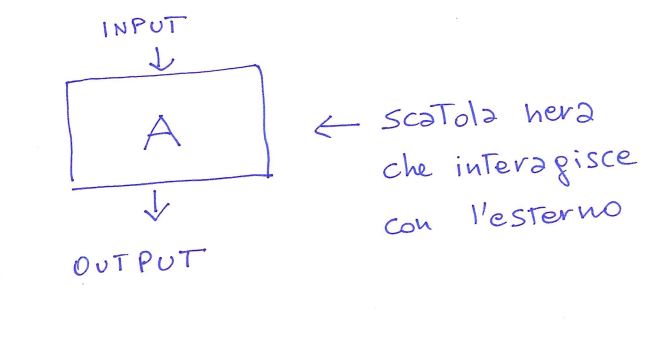
Un percorso diventa una parola (sequenza ordinata di simboli).

Grazie all’automa posso studiare a tavolino problemi legati al sistema

* Individuare una parola che mi porta da a ;
* Individuare una parola che mi porta da a passando attraverso (o evitando di passare attraverso ) ;
* Individuare tutte le parole prive di cicli che mi portano da a ;
* Tra queste ultime individuare la parola più corta.

Automi a stati finiti come riconoscitori di linguaggi formali

In prima approssimazione si presenta così:



Un esperimento consiste in:

* passare in input ;
* osservare l’uscita, se essa sarà uguale a 1 allora mentre se sarà uguale a 0 allora .

Il comportamento di è dato dall’insieme dalle parole che accetta = Linguaggio accettato è un riconoscitore di linguaggi, in particolare gli automi a stati finiti riconoscono i linguaggi di tipo 3.

In dettaglio:

* In ogni istante di tempo l’automa si trova in un particolare stato. Inizialmente si trova in uno stato iniziale ;
* In funzione del simbolo letto e dello stato attuale cambia stato, quindi chiameremo la funzione di transizione:
  + stato prossimo di essendo in e leggendo ;
* Una volta letta l’intera parola , raggiunge uno stato e l’uscita dipende da :
  + o se è uno stato finale, funzione di uscita.

Formalizziamo:

Definizione

Un automa a stati finiti è una tupla

Dove:

alfabeto di input

insieme degli stati (se è finito allora è a stati finiti)

stato iniziale di partenza

funzione di transizione

funzione di uscita

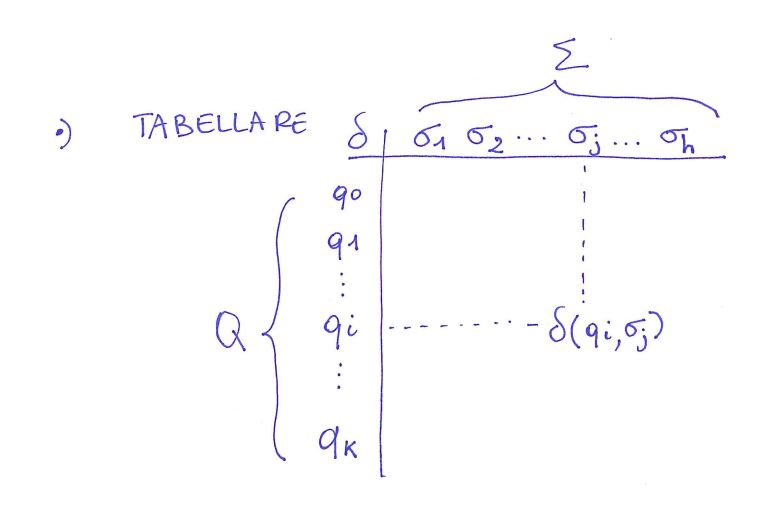
oppure

stati finali o accettanti

Osservazione:

* Data posso avere come ;
* Dato posso avere come se mentre sarà se

Rappresentazioni di :

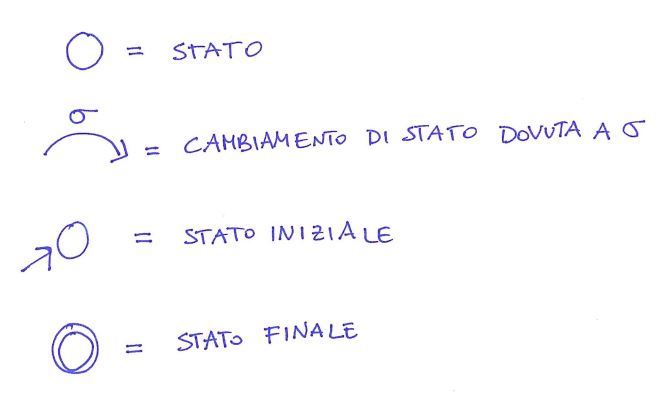


Esempio:

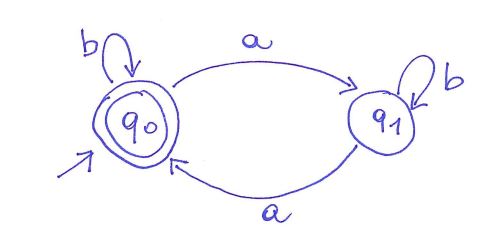
iniziale

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Diagramma degli stati



Esempio grafico:



Una parola è accettata se partendo da stato iniziale, induce un cammino che termina in uno stato finale.

Esempi di parole:

è accettata

è rifiutata

è accettata, se fosse stato sarebbe stata accettata solo con pari

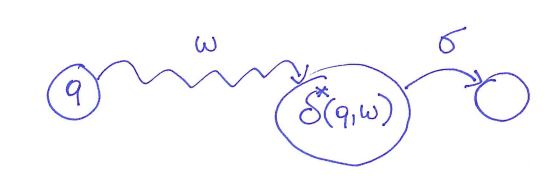
Formalizzo il concetto di linguaggio riconosciuto

Introduciamo la che è la estesa alle parole:

Definizione induttiva:

| { |  |
| --- | --- |
|  |

dove e



linguaggio riconosciuto da

Esempio:

numero di in

Lezione del 26 Aprile 2022

Automi a stati finiti

Li indichiamo con

Dove:

* è l’alfabeto ;
* è l’insieme degli stati ;
* è lo stato iniziale ;
* è la funzione per la transizione tra stati, essa ha due parametri prendendo uno stato e un simbolo e restituisce uno stato, quindi e quindi restituisce lo stato successivo a dopo aver letto il simbolo ( con è lo stato raggiunto da dopo aver letto la parola ) ;
* insieme degli stati finali ;
* output degli stati finali .

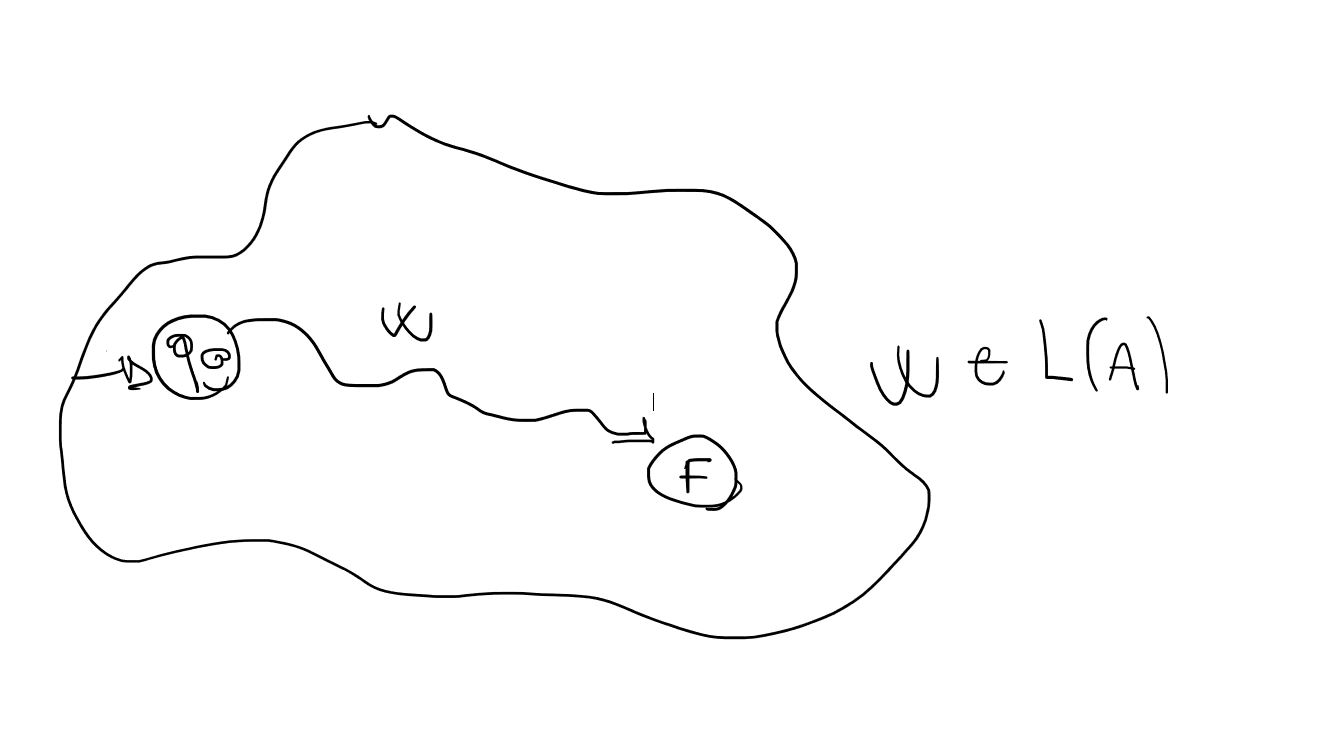
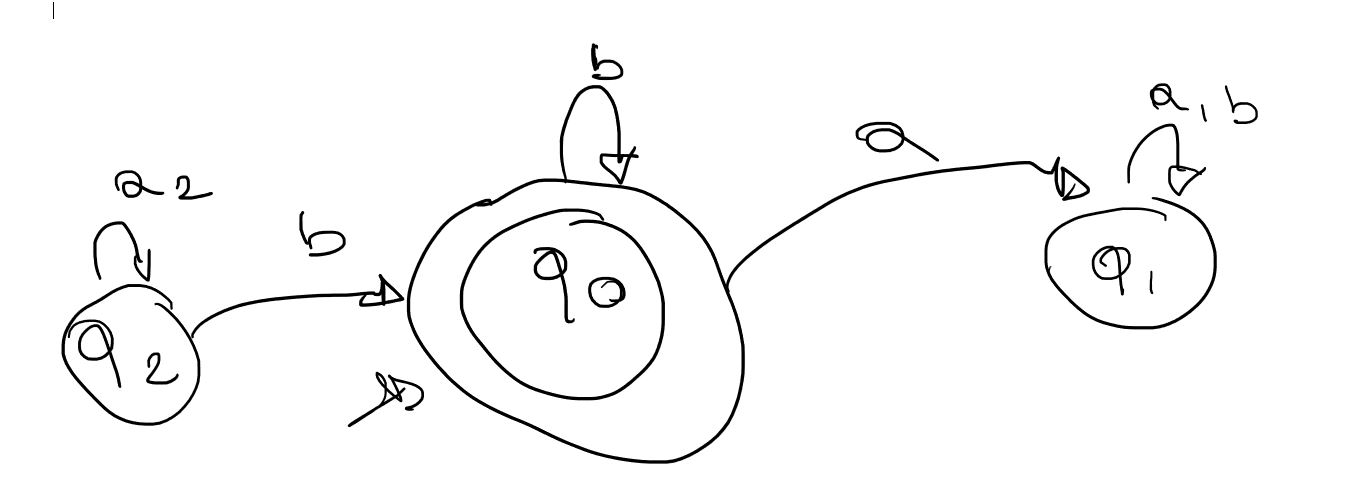


Diagramma degli stati di

Stati particolari

* Stato trappola: è uno stato trappola se vale che e (di solito assorbe le parole rifiutate) ;
* Stato osservabile: è uno stato osservabile se (è osservabile se è raggiungibile dallo stato iniziale) .

Esempio:



Si può notare che è uno stato osservabile grazie ad ed è anche uno stato trappola perchè non esce da e non è uno stato finale.

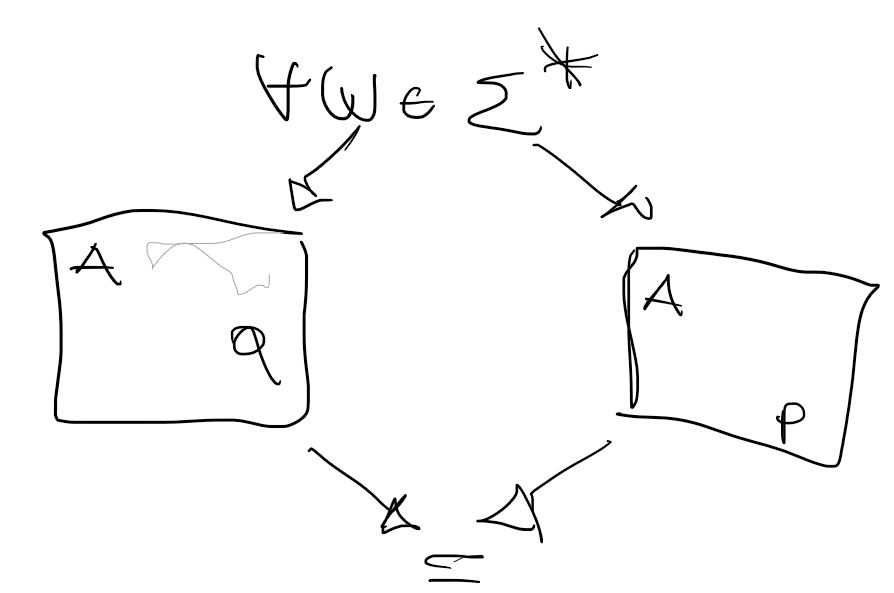
Lo stato è osservabile partendo sempre da con la parola o con la parola vuota () , ma non è uno stato trappola perchè è finale e con vado in .

Lo stato è non osservabile perchè da non è raggiungibile (questo stato si può eliminare).

Definizione

Stati indistinguibili

e si dicono indistinguibili quando



Stati distinguibili

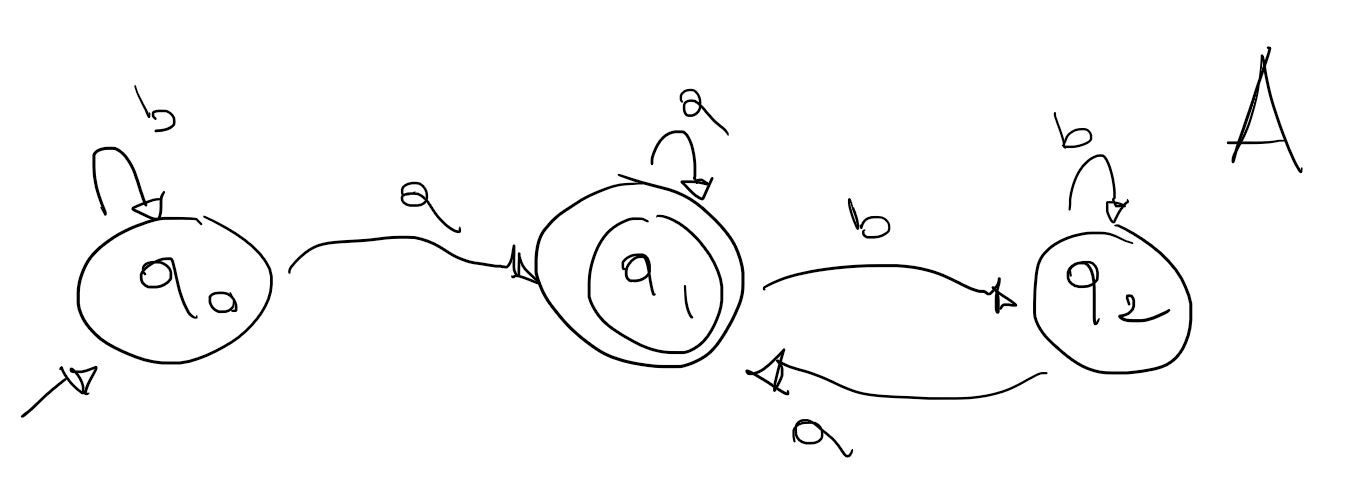
e si dicono distinguibili quando (quindi sono distinti per almeno una parola).

Notazione

indistinguibili

distinguibili

Esempio



non posso eliminare alcuno stato

(nello stati finale entrano le frecce etichettate con )

(tutte le frecce etichettate con mi portano in stati che non appartengono a )

Considero la coppia e mi chiedo se sono indistinguibili.

Leggendo vanno entrambe nello stato finale, leggendo vengono mandate entrambe in uno stato però sono distinguibili per dato che viene rifiutato mentre va in uno stato finale.

Vale lo stesso per .

E per per la parola vanno entrambe in mentre per e per vengono rifiutate.

Come sfruttare questa relazione per ridurre gli stati dell’automa

: relazione binaria sull’insieme

Proprietà:

* Riflessiva ;
* Simmetrica con ;
* Transitiva ;
* se vale che ;

(se parto da due stati indistinguibili e leggo una qualsiasi parola raggiungo 2 stati indistinguibili)

relazione di equivalenza

Quindi essendo una relazione di equivalenza indica una partizione su ( classi di equivalenza di stati)

Partizione in classi:

* ;
* ;
* .

Dato

Posso costruire che ha come stati le classi di equivalenza di

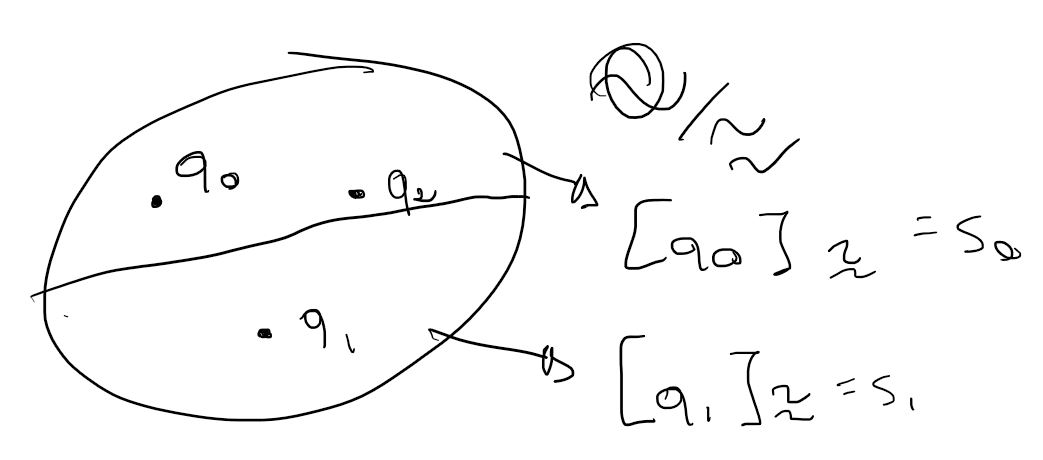
Formalmente

Dove:

* ;
* definito come .

Esempio:

Prendiamo l'automa precedente e costruiamo



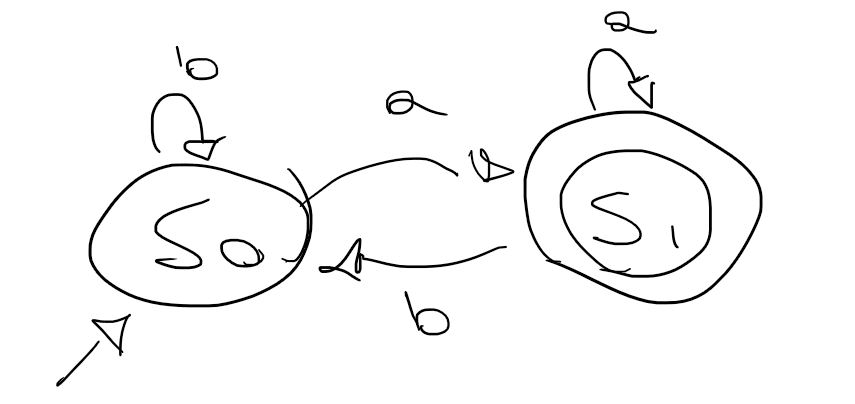
stato iniziale

da costruire

Definiamo :

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Vediamo il diagramma degli stati di



è equivalente ad ma cambia il numero di stati stati e stati.

Nella prossima lezione vedremo stati minimi e massimi per un automa.

Lezione del 28 Aprile 2022

Problema di sintesi degli automi

Abbiamo in input un linguaggio e dobbiamo trovare un automa per . Per questo automa ricaveremo automa massimo e attraverso due metodi successivamente ricaveremo l’automa minimo.

In particolare vedremo:

1. Com’è fatto l’automa massimo per ;
2. Come ottenere l’automa minimo per .

Automa massimo = automa con il maggior numero di stati possibili.

Automa minimo = automa con il minor numero di stati possibili.

Dall’automa massimo otteniamo l’automa minimo (nostro obiettivo).

Punto 1

Automa massimo:

Desideriamo che quando leggo una parola l'automa mi porti in uno stato diverso. L’idea è: per ogni parola in input raggiungo uno stato diverso.

(ogni parola ha il suo stato)

Inoltre, per non perdere stati, tutti gli elementi di devono essere osservabili.

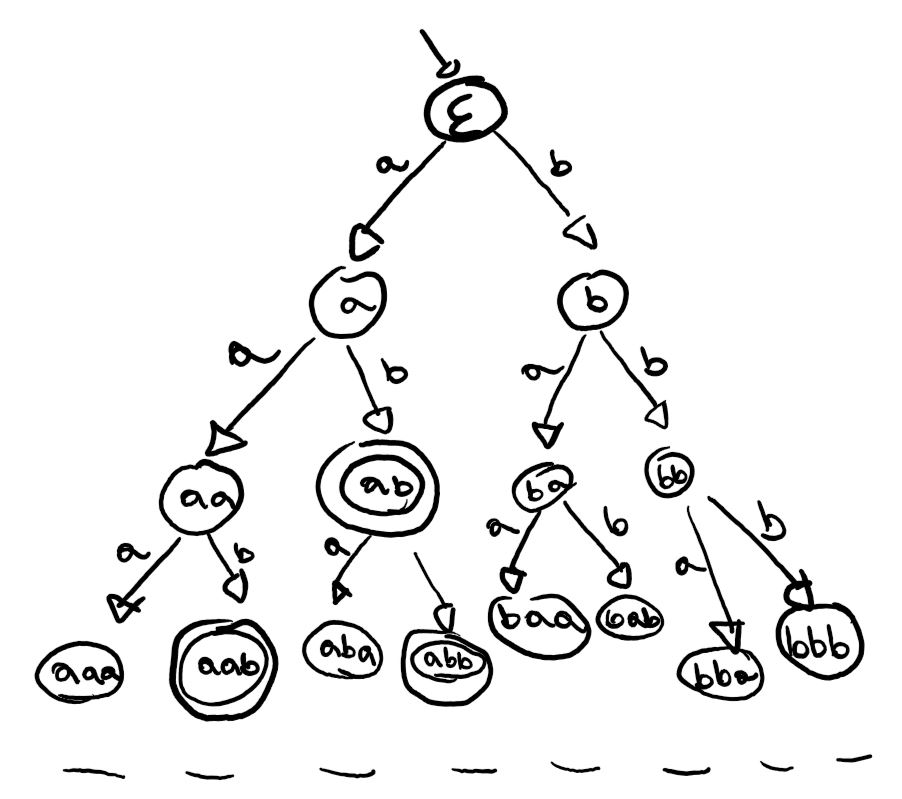
(definizione di stato osservabile)

Definizione

dove

etichetta dello stato iniziale come la parola vuota

Vediamo un esempio di automa massimo per il linguaggio



Si può continuare potenzialmente all’infinito.

Punto 2

Automa minimo:

Abbiamo due tecniche per trovare questo automa:

1. Si usa l’automa massimo (introdotto solo ai fini dell’automa minimo) ;
2. Si usa un automa a stati finiti per e poi si trova quello equivalente.

Prima tecnica:

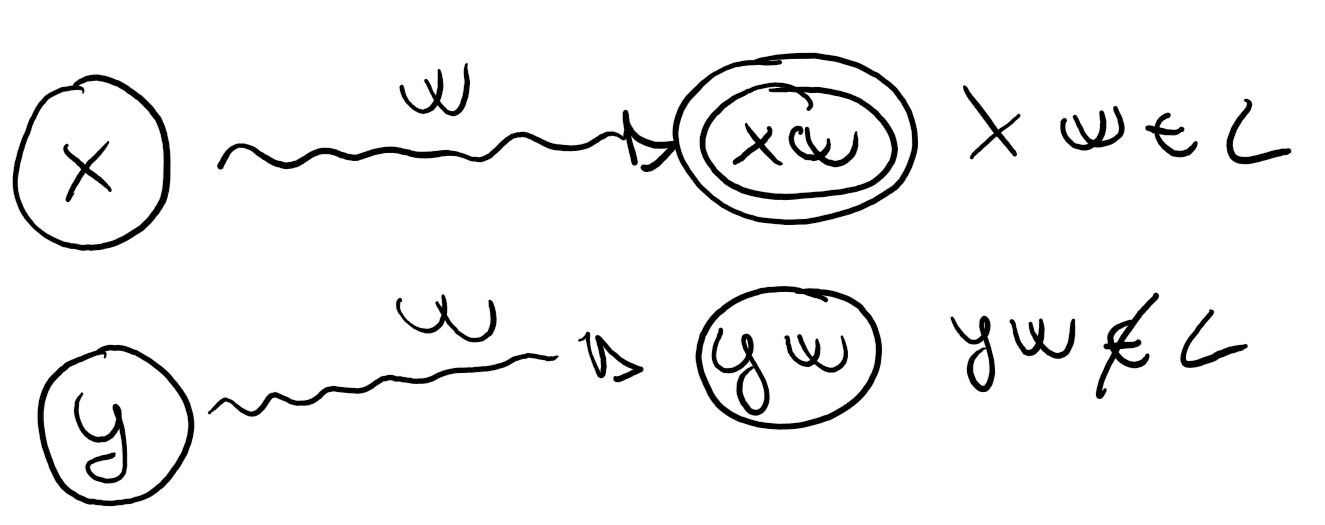
Teorema

è l’automa minimo per . Cioè .

Quindi devo mostrare che se ho un automa per allora (numero di stati di è maggiore del numero di stati di ).

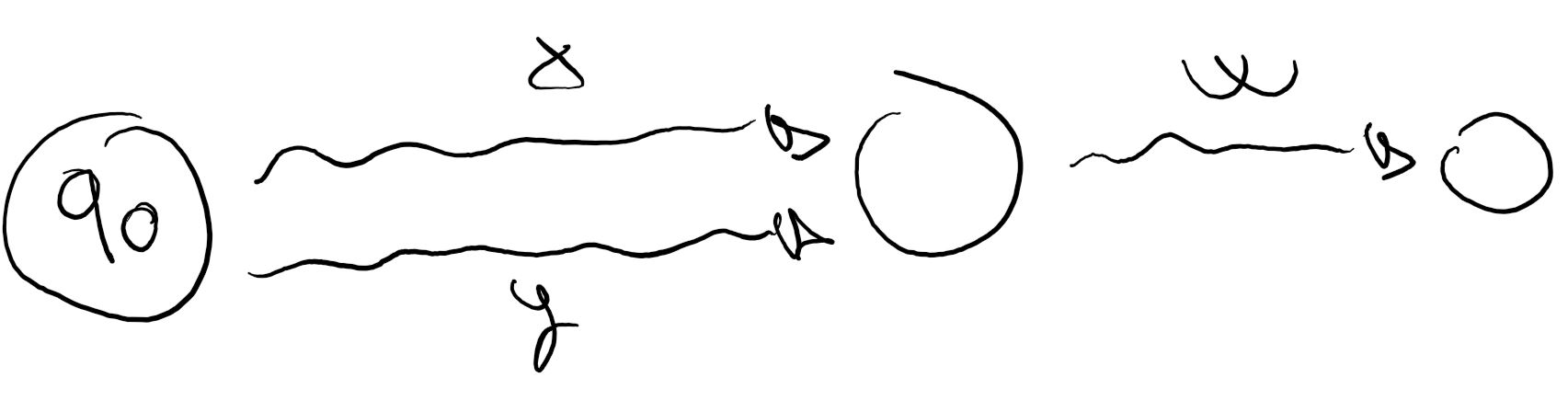
Per prima cosa mostriamo un fatto.

FATTO: gli stati distinguibili in sono stati distinti in . Infatti siano e due stati di distinguibili in esso, cioè: :



Allora in :

altrimenti



Contraddizione perchè così abbiamo e appartengono o non appartengono entrambi a (quindi l’automa non li riconosce).

Ora

Siano gli stati di , allora gli stati sono stati distinguibili in . In virtù del precedente fatto (appena dimostrato) sono stati distinti in (dove è un generico automa per ) ha almeno tanti stati quanti

e quindi è minimo per il linguaggio attraverso .

Seconda tecnica:

Teorema

Sia un automa a stati finiti per tale che gli stati di siano:

* tutti osservabili ;
* tutti distinguibili .

Allora è minimo per .

Corollario

è minimo per se ha tutti gli stati osservabili.

Dimostrazione del teorema

Siano gli stati di (dove sarà lo stato iniziale).

Vale che:

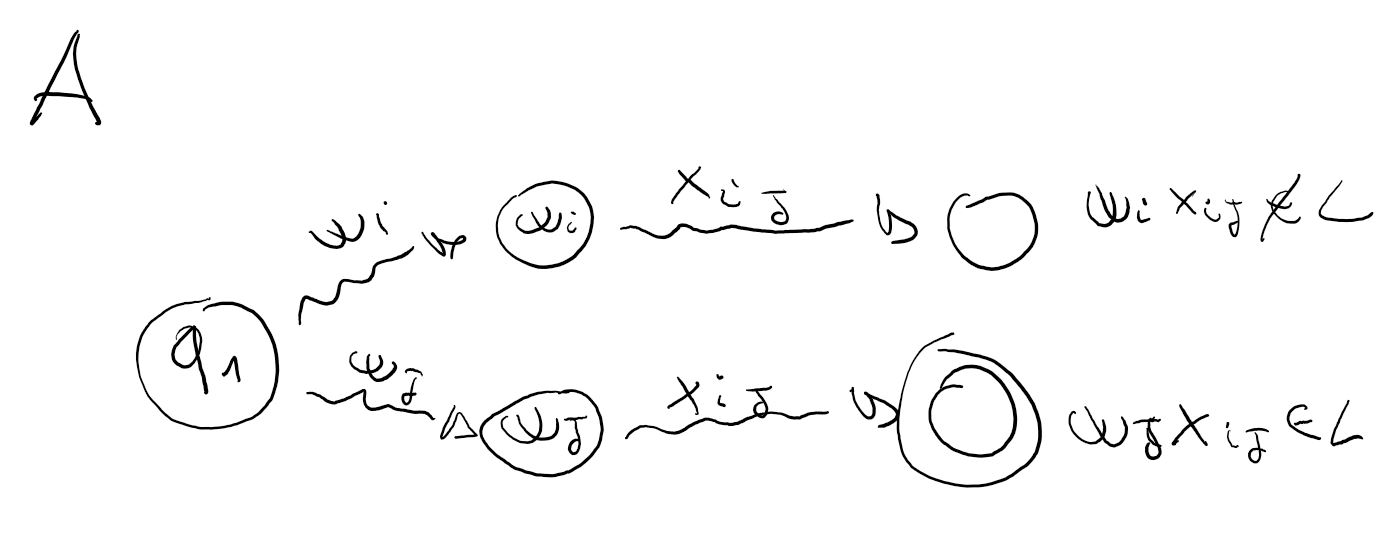
* sono tutti osservabili ;
* sono tutti distinguibili .

Sono osservabili:

Allora esistono parole tale che .

Sono distinguibili:

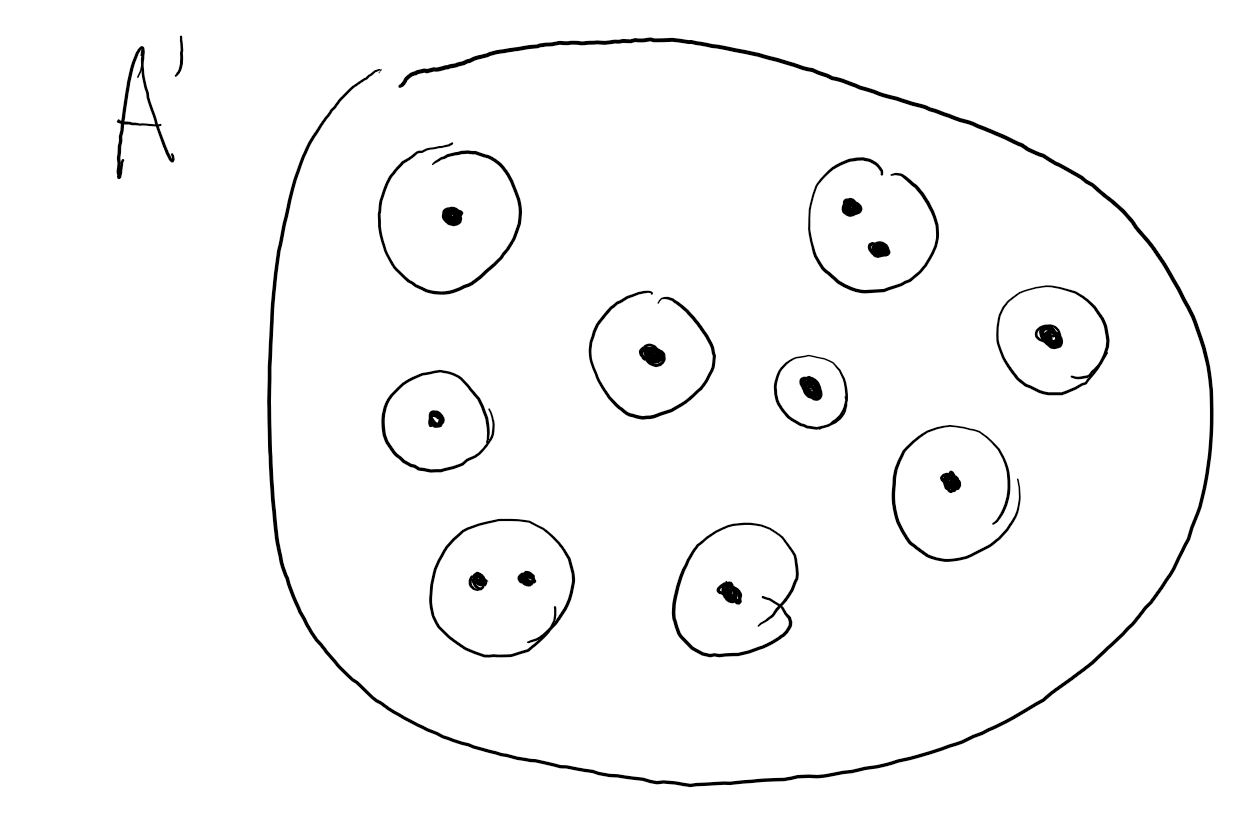
Allora esistono parole che distinguono gli stati



Sia un automa per con meno stati di (meno di numero di ).

(tecnica per assurdo)

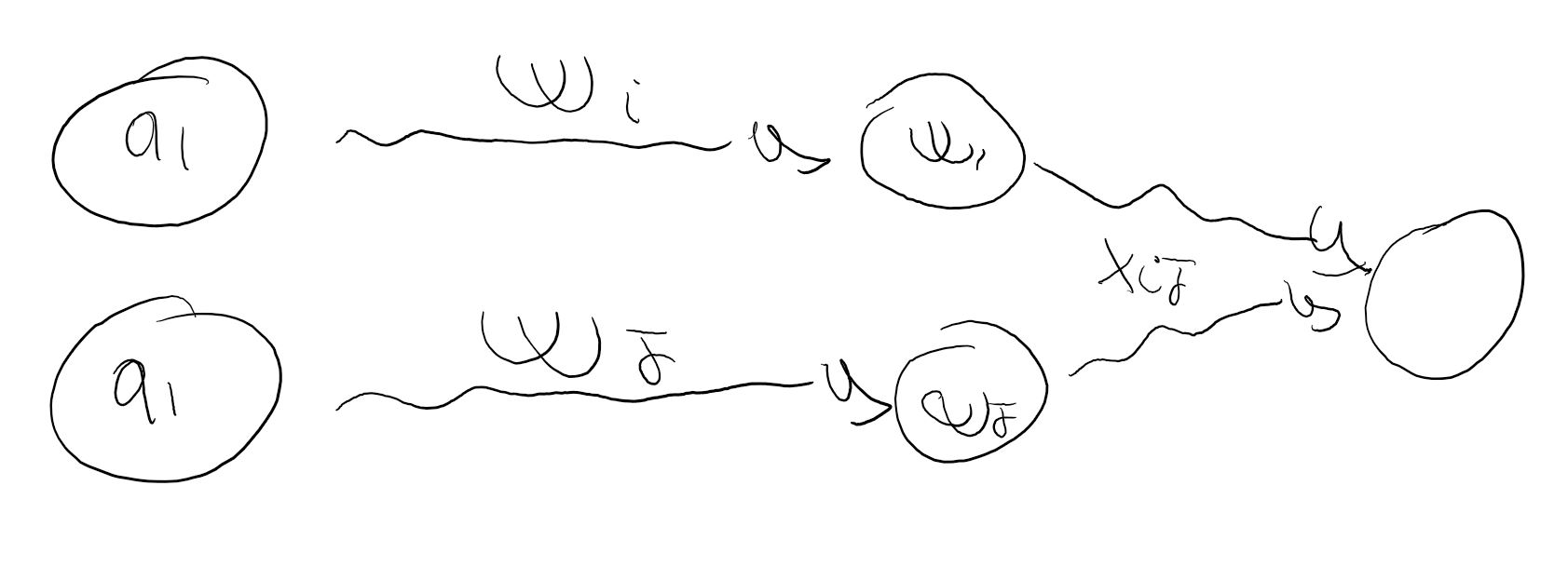
Diamo in input ad le parole (parole date in precedenza ad ) si ha



Dato che ho meno stati, due parole raggiungono lo stesso stato dato che

Secondo il principio della piccionaia (se abbiamo più piccioni che gabbie avremo almeno una gabbia con due piccioni).

Siano e queste parole che mi portano nello stesso stato. In dunque accade che:



Così ottengo che e sono entrambe o quindi non riconosce parole in (Assurdo!!!) e infine non esiste in automa in con meno stati di

Lezione del 3 Maggio 2022

Algoritmi di sintesi ottima di automi

Sintesi ottima di automi

* Dato un linguaggio , trovare l’automa minimo per .

Abbiamo due tecniche:

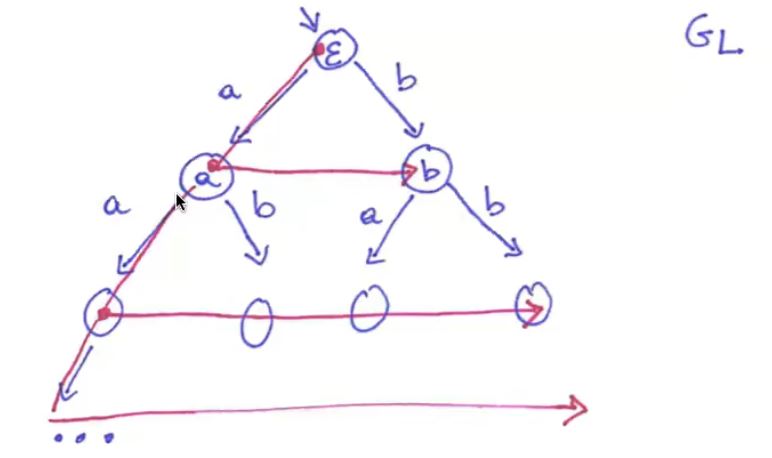
1. Se ho per ( ha stati finiti):
   1. eliminare gli stati non osservabili ;
   2. costruire ;
2. Se non ho per :
   1. costruisco automa massimo ;
   2. costruisco .

Algoritmi

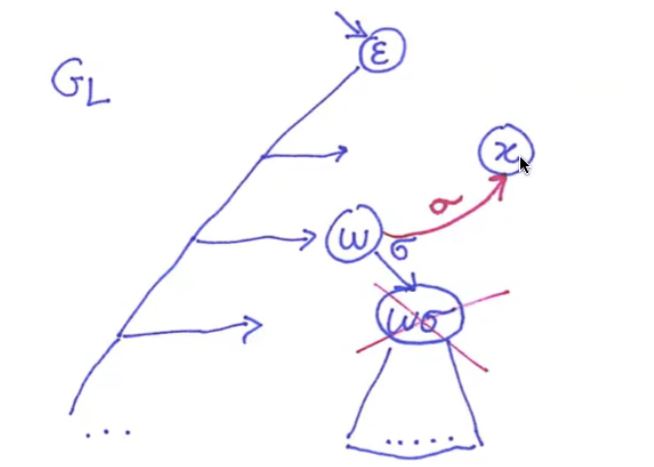
1. E’ possibile confrontare gli stati seguendo un ordine casuale ;
2. Devo applicare un algoritmo sui confronti ben preciso .

Algoritmo per

* Si parte dalla radice e si visitano i nodi (stati) in ampiezza ;



* La visita di un nodo consiste nel confrontare il nodo attuale con tutti quelli precedenti, verificando se sono indistinguibili oppure no ;
* Sia (con ) il nodo attuale. Se accade che è indistinguibile dal nodo allora:
  + cancello e il suo sottoalbero ;
  + resta pendente l'arco uscente da etichettato con . Ridireziono l’arco verso .



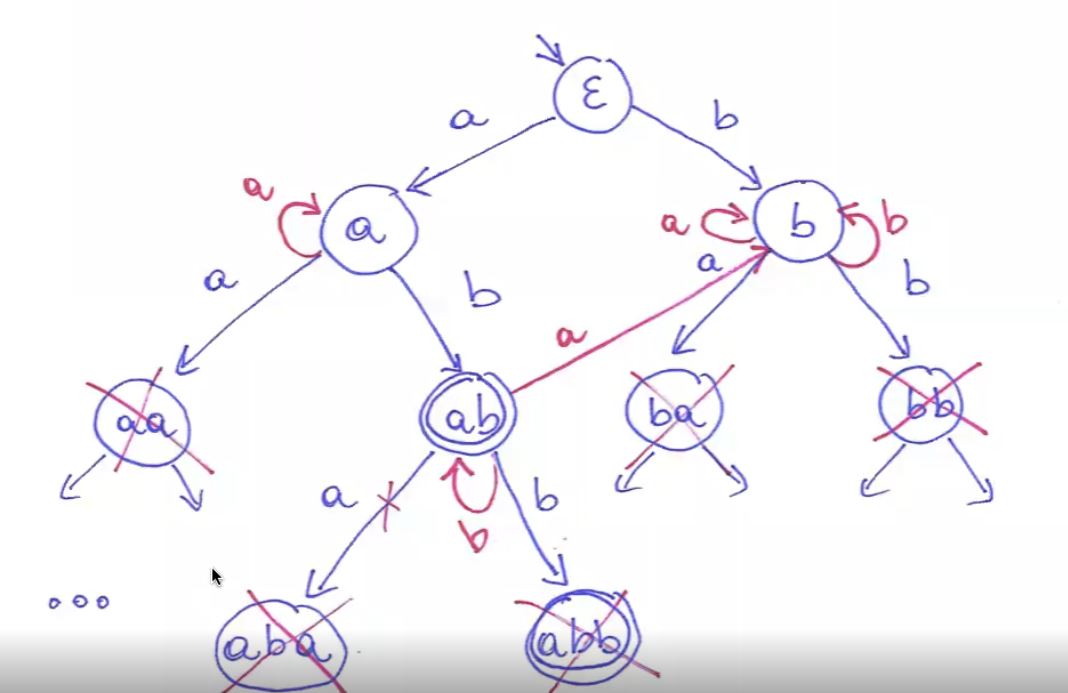
Osservazioni:

* Lo stato iniziale resta ;
* Gli stati finali sono i sopravvissuti etichettati con parole di .

Due prove di esecuzione:

* Per otteniamo un automa a stati finiti: per il teorema dimostrato ;
* Per otteniamo un automa con infiniti stati, conseguenza: non è un linguaggio regolare.

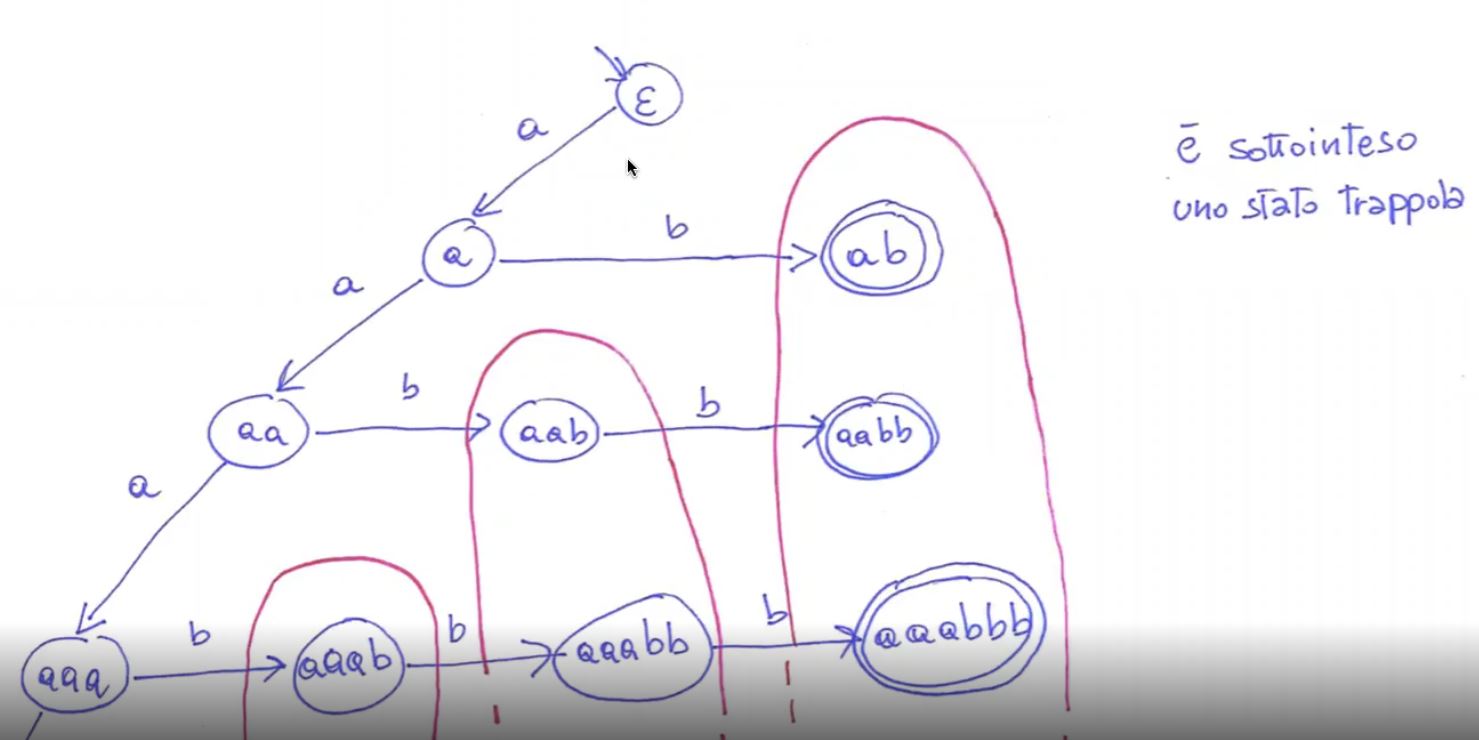
Costruzione di per con correzioni secondo l’algoritmo



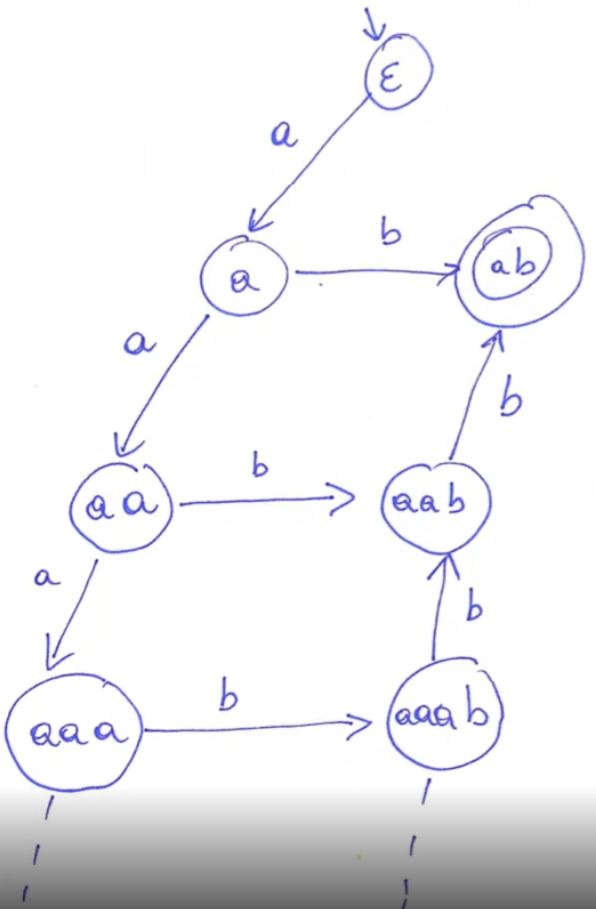
Applicazione dell’algoritmo. Visite:

* resta ;
* :
  + a causa della parola (partendo da , viene accettata ma non accade il viceversa);
* :
  + a causa della parola (partendo da , viene accettata ma non accade il viceversa);
  + a causa della parola (partendo da , viene accettata ma non accade il viceversa) ;
* :
  + a causa della parola (partendo da , viene accettata ma non accade il viceversa) ;
  + indistinguibile perchè le parole lette da entrambi gli stati vengono rifiutate o accettate allo stesso modo, quindi cancello lo stato ed il suo sottoramo e collego l’arco allo stato indistinguibile precedente ;
* :
  + è inutile confrontare uno stato finale con gli stati non finali, dato che ci sarà sempre la parola a differenziarli ;
* e :
  + sono due stati che vengono rifiutati perchè partono da , sono indistinguibili da quindi essi possono essere eliminati e i loro archi spostati verso di esso ;
* :
  + viene rifiutata a prescindere, quindi è indistinguibile da (stesso procedimento di prima) ;
* :
  + stato indistinguibile con (in quanto finale) e in quanto con le parole vengono accettate mentre con vengono rifiutate in entrambi gli stati .

Costruzione di per



(Sono stati rimossi gli stati trappola, e quindi abbiamo solo i cammini per gli stati terminali)



Da questo grafico capisco che ha stati infiniti e quindi non è regolare. Avremo quindi un ramo infinito con ed un ramo infinito con .

FATTO

non è regolare

Dimostrazione

* L’automa minimo per ha infiniti stati ;
* Allora non ammette un automa a stati finiti ;
* vista l’equivalenza tra automi finiti e grammatiche di tipo 3, esso non è regolare.

Lezione del 5 Maggio 2022

Teorema

è generato da di tipo 3 è riconosciuto da a stati finiti

Dimostrazione 1

Da ricavo

Sia , posso definire

(insieme dei simboli terminali)

(insieme delle variabili)

(assioma)

(regole di produzione) :

* (per ogni stato finale dell’automa) ;
* (posso mettere una regola di questo tipo se e solo se dando a ottengo , quindi nel diagramma degli stati, per ogni arco possiamo inserire una regola di questo tipo).

Correttezza della costruzione

Proprietà:

si dimostra per induzione

Caso base:

Vero

La prima situazione è sempre vera nell’automa mentre la seconda è vera se consideriamo zero passi di derivazione

Passo induttivo:

Suppongo vera la proprietà per e lo dimostro per .

Si consideri , con e , quindi

e (la prima per ipotesi e la seconda per costruzione) e

Dimostrata

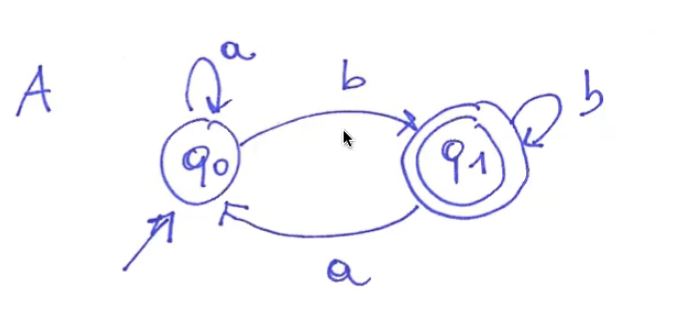
Adesso facciamo vedere l’equivalenza:

e

e

quindi infine

Esempio:



da ricavo di tipo 3:

, , l’assioma è ,

* ;
* ;
* ;
* ;
* .

esempio

Dimostrazione 2

Da di tipo 3 ad :

* si inverte la costruzione precedente ;
* ma deve essere nel formato:
  + ;
  + ;

Data definisco :

simboli terminali

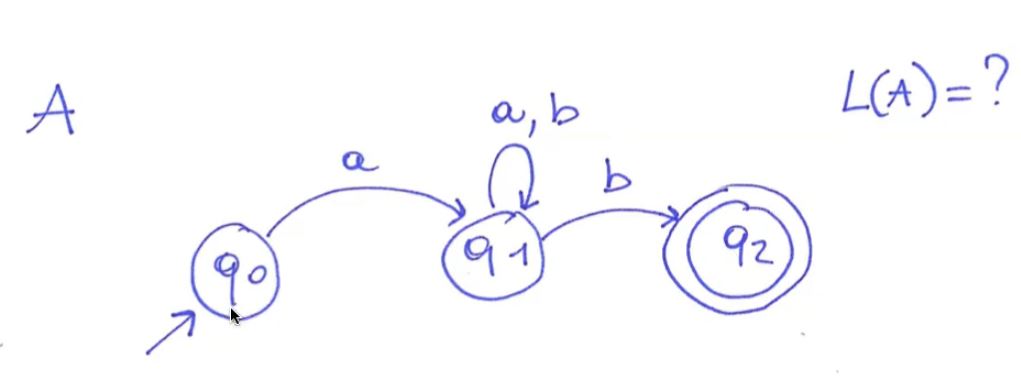
variabili

assioma

è così definita:

Esempio:

, , , ,



NOTA: non può essere più considerata una funzione, quindi non è un automa DETERMINISTICO (DFA) ma è NON DETERMINISTICO (NFA)

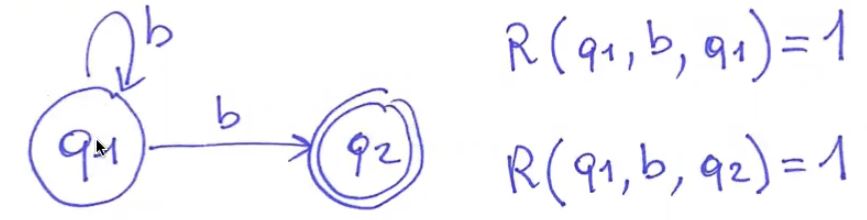
Automi a stati finiti non deterministici (NFA)

Definizione

Un NFA è un sistema

dove è la relazione di transizione

esempio



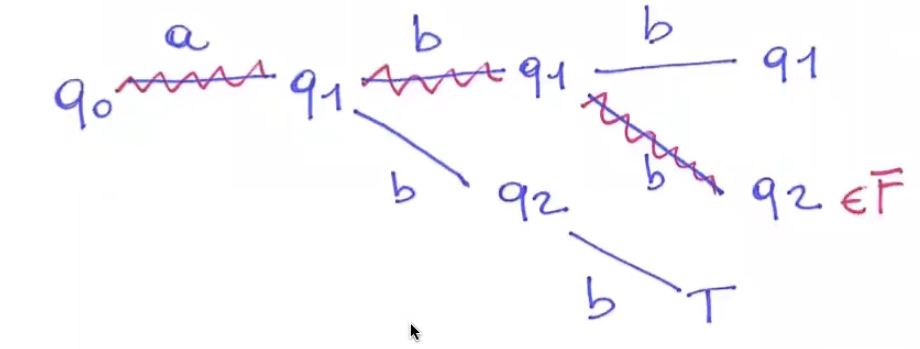
Interpretazione:

sceglie non determinatamente di transitare in o in

Osservazione:

Data una parola in input ad , è possibile che induca più di un cammino possibile

Esempio



Dove è uno stato trappola

Definizione

Linguaggio riconosciuto da un

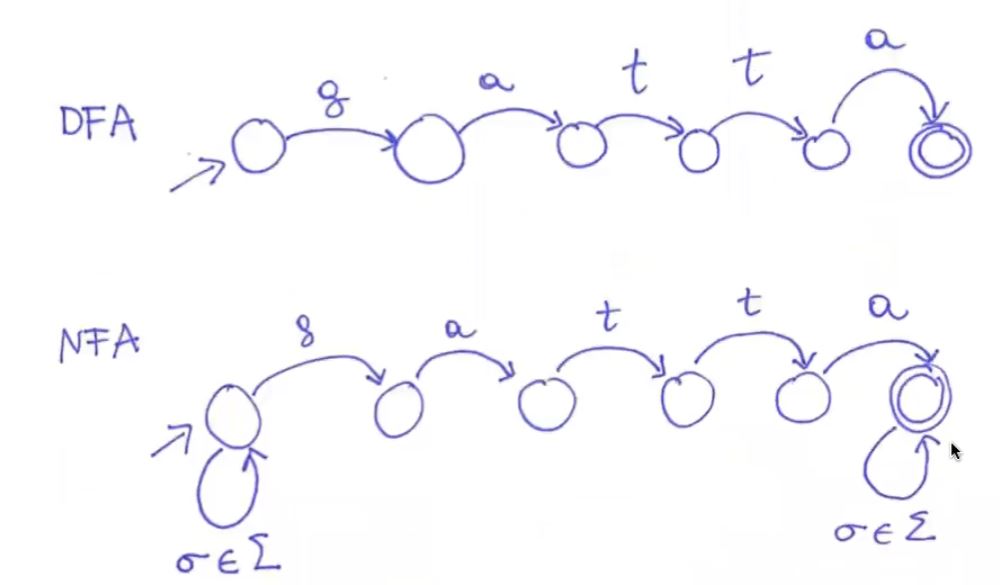
* è accettata se ammette un cammino che porta in uno stato finale ;
* il linguaggio riconosciuto è dato dalle parole che soddisfano il criterio di accettazione.

Lezione del 10 Maggio 2022

Semplici applicazioni:

* con un riconosco facilmente una parola ;
* con un riconosco testi che contengono una certa parola.

Esempio:



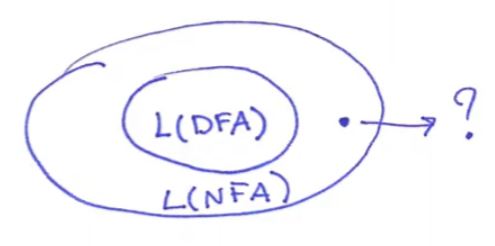
(Il secondo riconosce interi testi che contengono la parola)

Esecuzioni:

* “La gatta” è accettata ;
* “Il gatto” è rifiutato (una volta arrivati al secondo stato “t” vengo portato in uno stato trappola) ;
* “Il gatto della gatta” è accettato.

la classe dei linguaggi accettati da

la classe di linguaggi accettati da



Nota: I sono un caso particolare di

Esiste tale che è accettato da un , ma non esiste un che lo riconosce?

La risposta alla seguente domanda è NO!

In realtà entrambi i tipi di automi sono equivalenti

Teorema

Per ogni riconosciuto da un esiste un che lo riconosce (è equivalente)

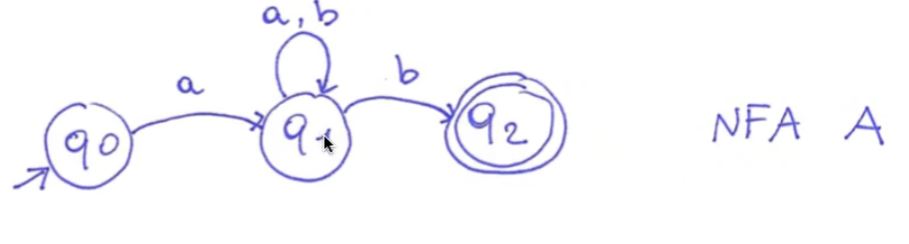
Corollario

Dimostrazione del teorema

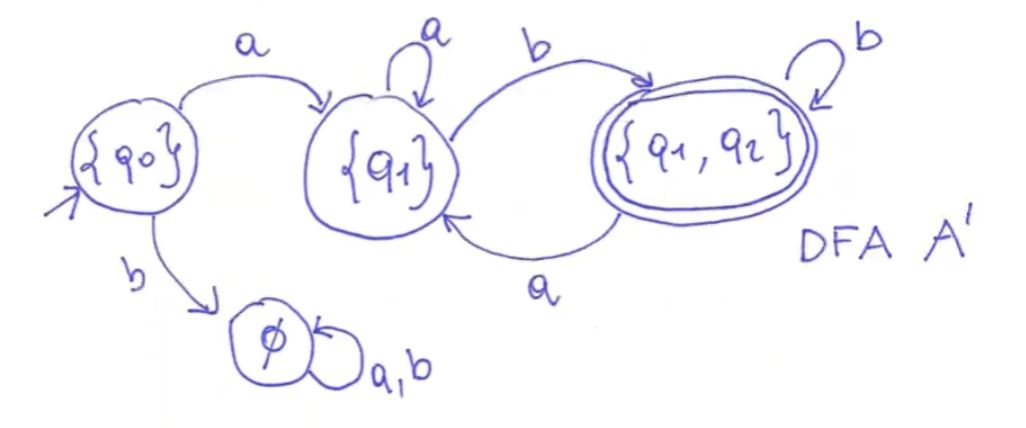
Dato costruisco deterministico (l’insieme dei sottoinsiemi di ) dove:

che soddisfa (Prendo prima un elemento di , sottoinsieme di , poi cerco tra i suoi elementi gli stati raggiunti dai propri elementi)

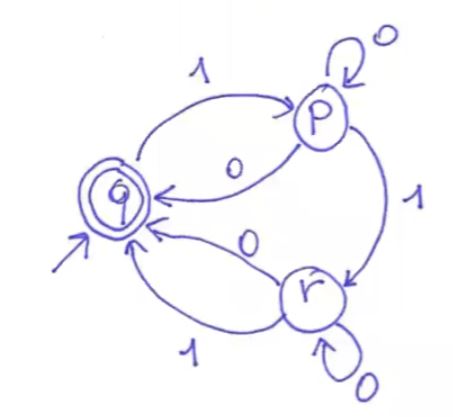
Esercizio



(Quando manca la transizione nel primo automa in teoria andremo in uno stato trappola, per convenzione poi lo chiameremo come l’insieme vuoto)

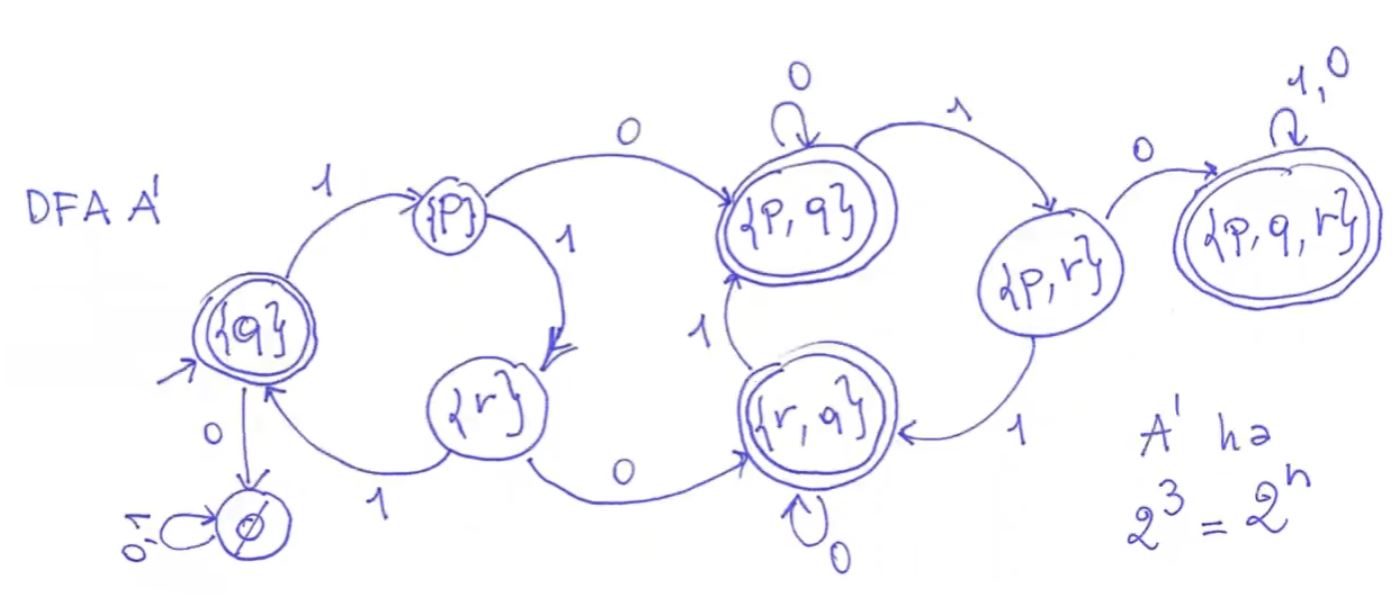


Trasformazione con incremento esponenziale degli stati

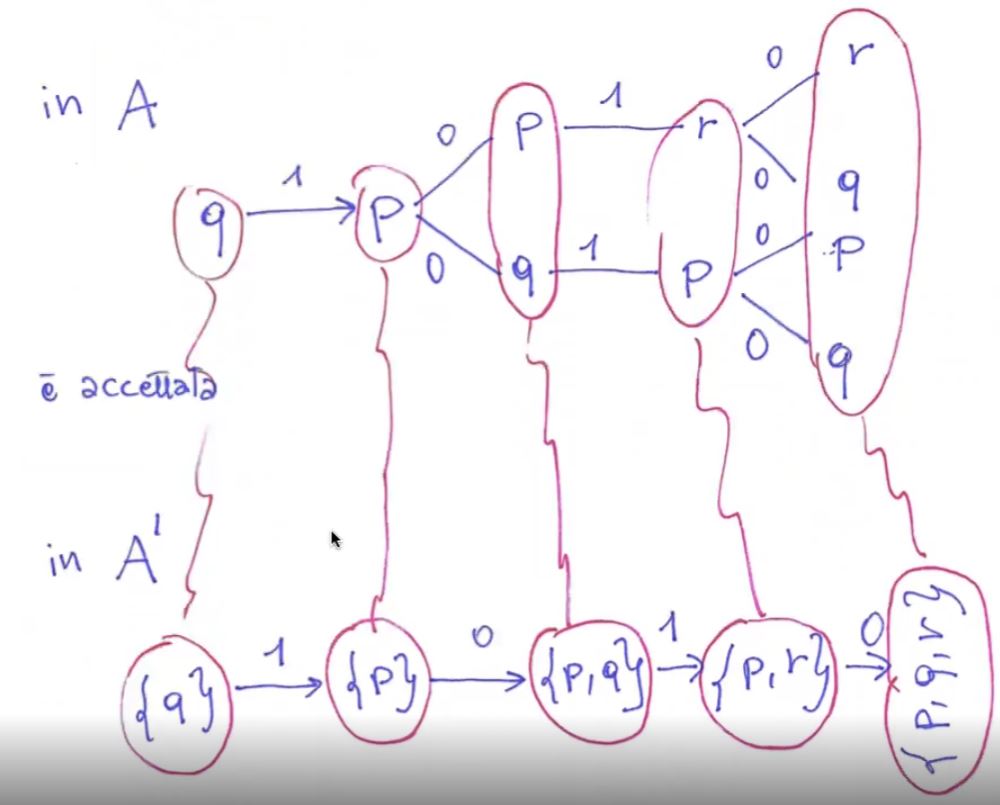


il quale ha stati

(Questo automa si potrebbe ingrandire all’infinito)



Input: 1 0 1 0



Viene accettata

Nota: Data il percorso nel indotto da simula tutti i cammini possibili di nell’

Lezione del 12 Maggio 2022

Altro formalismo per i linguaggi regolari

Espressioni regolari (ER)

Questo è un altro formalismo per definire questi linguaggi, essi sono espressioni fatte di simboli e operazioni che servono per denotare un linguaggio. Ci sono le operazioni di prodotto e di (Chiusura di Kleene).

Definizione

(Forma induttiva)

Una espressione regolare su è:

* ;
* ;
* .

(ER base)

Se e sono espressioni regolari allora:

* (unione);
* ;
* .

(ER complesse)

Sono espressioni regolari.

Le ER denotano dei linguaggi:

|  | denota | Linguaggio |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Date le ER che denotano e allora:

|  |  |
| --- | --- |
|  |  |
|  |  |

Esempi:

* denota ;
* denota
* Identificatori di variabili, devono rispettare una serie di regole, in questo caso abbiamo scelto la regola “non si può iniziare il nome della variabile con una cifra” (con “,”=”+”) ;
* Importo in euro .

Teorema di Kleene

è denotato da una ER è riconosciuto da un

(Dimostrazione costruttiva)

Dimostrazione 1

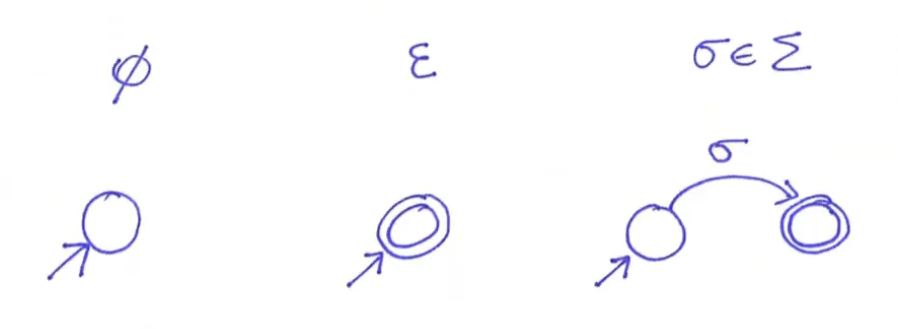
Data un'espressione regolare, creiamo un automa a stati finiti deterministico che riconosce tale linguaggio, attraverso una dimostrazione per induzione.

denotato da ER ammette un

Tecnica: per induzione

Caso base

Esistono per ER base



Nell’automa a stati finiti per il linguaggio vuoto abbiamo un solo stato, che qualsiasi cosa gli venga dato come input, non viene accettato, per i simboli finisce in uno stato trappola mentre per rimane in uno stato non finale.

Nel secondo caso avviene lo stesso per i simboli, mentre con epsilon si rimane nello stato iniziale che in questo caso è anche stato finale.

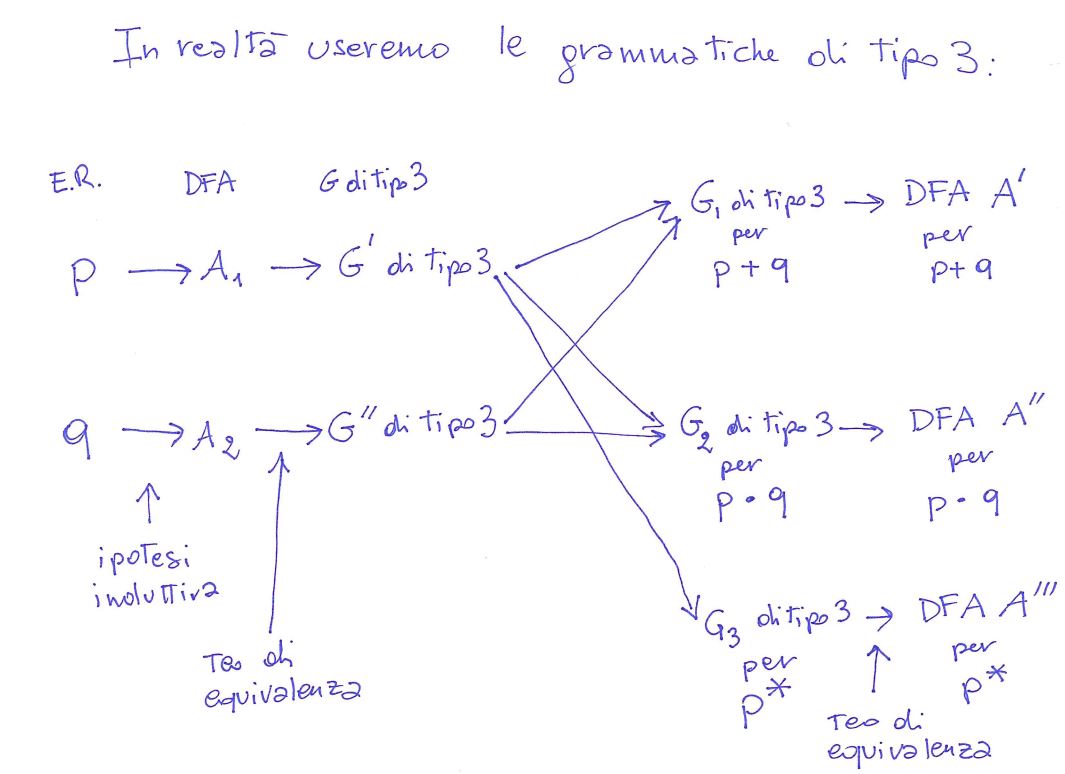
Nel terzo caso abbiamo uno stato iniziale e finiamo in uno stato finale unicamente con il simbolo dell’alfabeto (per via dell’arco etichettato con il suddetto simbolo).

Passo induttivo

Se esistono per ER e allora dimostro che esistono per:

* ;
* ;
* .

In realtà useremo le grammatiche di tipo 3



Per dimostrare questo schema dobbiamo far vedere come si compongono per poi ricavarne gli automi .

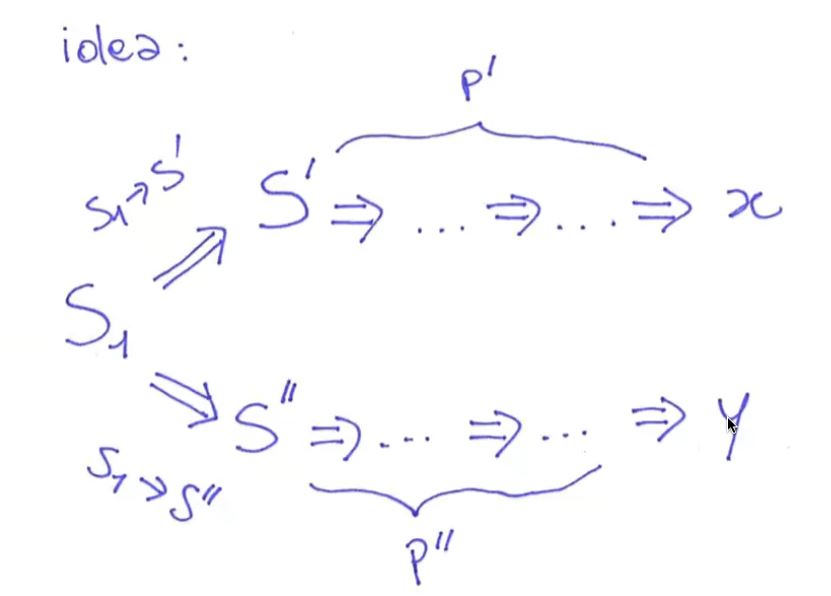
Abbiamo 2 grammatiche di tipo 3 (in una forma particolare):

NO

che generano

(caso particolare di queste due grammatiche)

1. Costruisco per

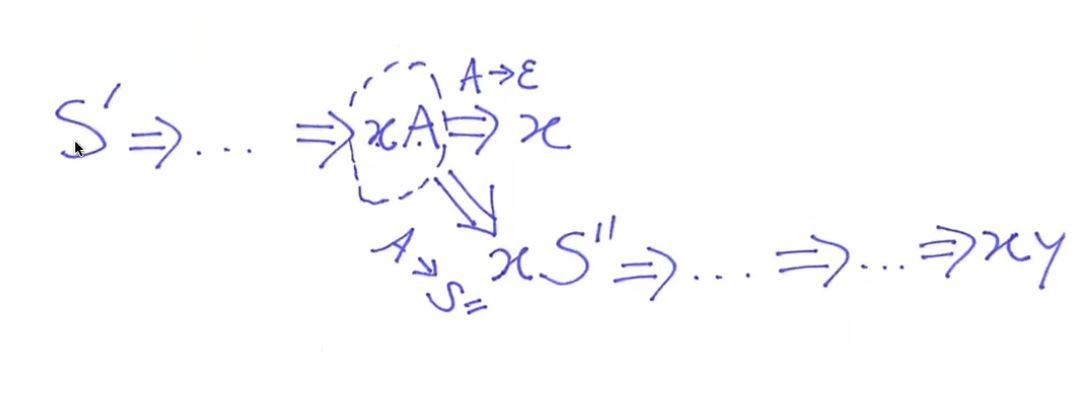


Richiesta

è di tipo 3?

Si, è lineare a destra:

1. Costruisco per



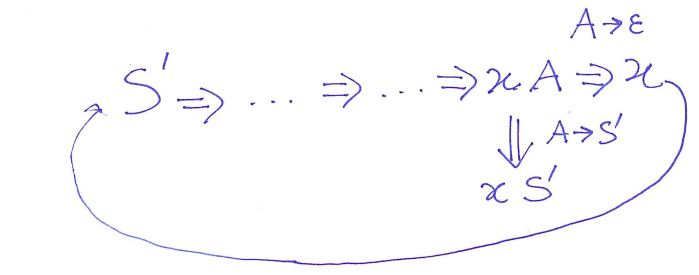
Esempio:

Adesso creiamo

è di tipo 3?

Si!

1. Costruisco per



Quella che abbiamo costruito è una grammatica per

Se allora dovremo aggiungere

Lezione del 17 Maggio 2022

Seconda parte della dimostrazione del teorema di Kleene

è denotato da un’espressione regolare è riconosciuto da un automa a stati finiti

Dimostrazione di: da un a una ER

Dato il , sia allora posso associare ad i seguenti :

,

,

,

…

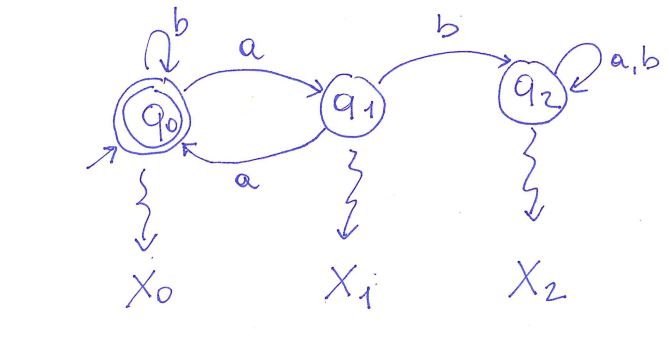
(Gli altri elementi sono uguali, cambia solo lo stato iniziale, cambiando così il linguaggio riconosciuto)

* Ognuno di questi riconosce un linguaggio.

Sia il linguaggio riconosciuto da .

Pertanto

Esempio:



Vediamo subito che sia uno stato trappola e che grazie a per far si che si accetti una parola abbiamo bisogno che quando compare una , compaia al suo seguito un’altra . Se ci fosse una in mezzo finiremmo in uno stato trappola.

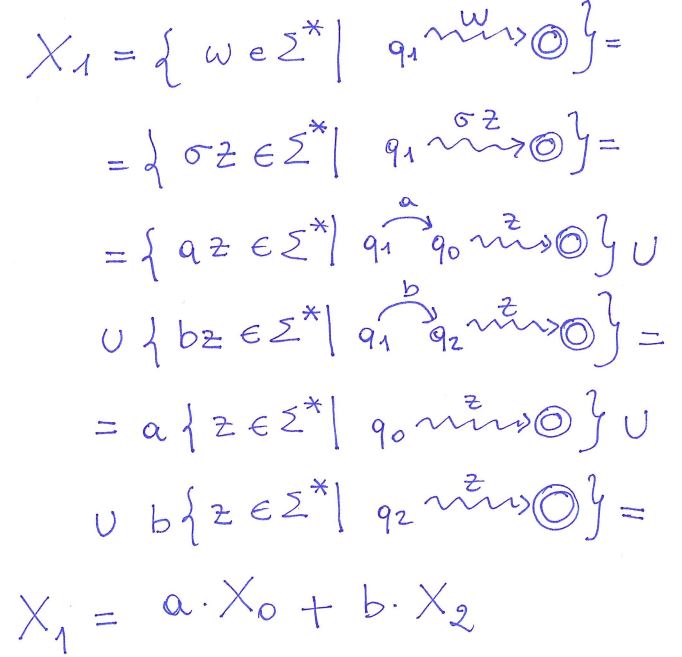
Non è facile, di base, elaborare un’espressione regolare dai seguenti stati. A questo automa associamo 3 stati, uno per stato (il quale diventa lo stato iniziale).

Il linguaggio che verrà riconosciuto da stato trappola sarà il linguaggio vuoto , però noi per trovarlo useremo una serie di procedimenti automatici.

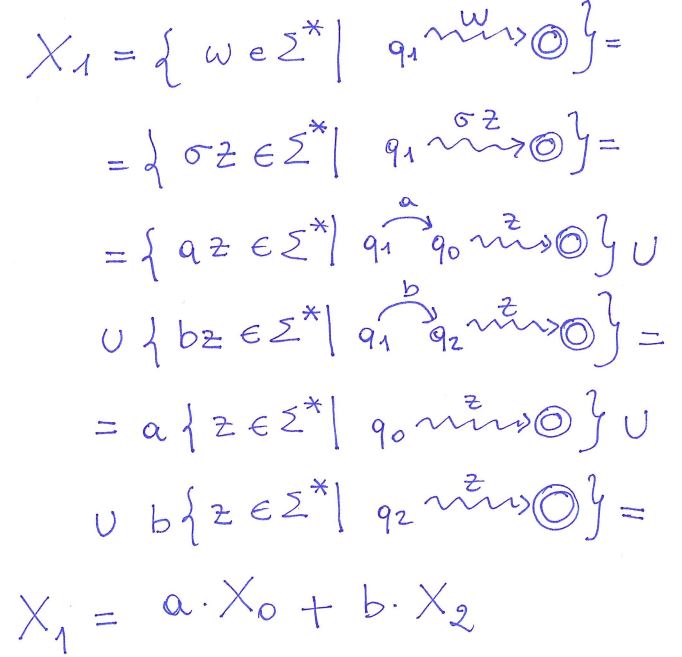
* Adesso cerchiamo di esprimere ogni linguaggio in funzione degli altri.

Per ottenere una ER per cerco di esprimerla in funzione delle altre.

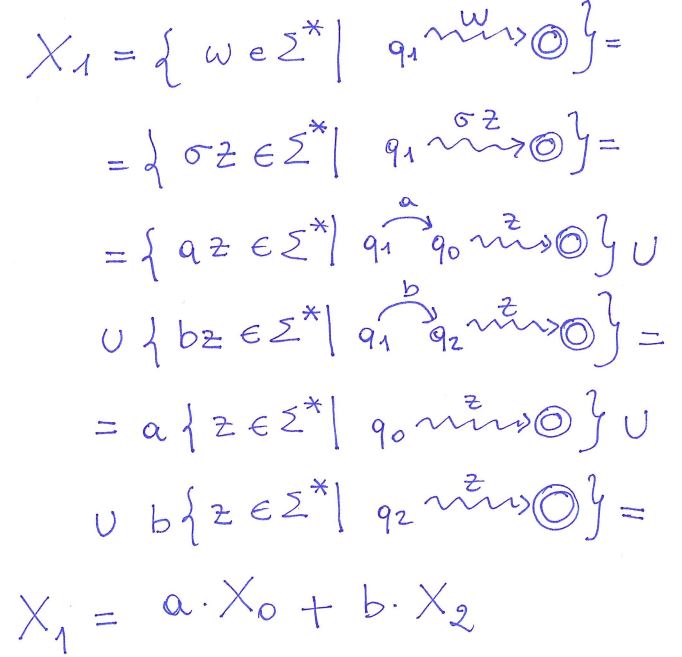
Esempio



Posso esprimere come dato che non può appartenere a (dato che non è finale). Questo può essere sia , sia .



La parola grazie ad andiamo nello stato , per poi attraverso arrivare ad uno stato finale (quindi in pratica è come se appartenesse al linguaggio ), lo stesso vale per .



In generale:

Il primo simbolo è una sommatoria, con un simbolo all’inizio con il linguaggio corrispondente allo stato raggiunto con il suddetto simbolo, ed infine abbiamo un solo se lo stato iniziale è uno stato finale.

Adesso per trovare l’espressione regolare dell’automa precedente mettiamo a sistema il risultato dell’applicazione della formula precedente nei tre linguaggi.

| { |  |
| --- | --- |
|  |
|  |

* Si ricava un sistema con (numero di stati di ) equazioni e incognite ;
* Per risolvere il problema devo ricorrere a equazioni del tipo: ( è l’incognita mentre e sono linguaggi) la cui soluzione è , se allora la soluzione è unica mentre se allora è la minima soluzione.

Verifica della soluzione

sostituisco ad :

Verificato!!!

Adesso che abbiamo verificato il risultato, lo applichiamo all’ultima equazione, la quale può essere riscritta così:

Dove e quindi

Elimino quindi una variabile dal sistema sostituendo l’espressione ricavata:

Dove e

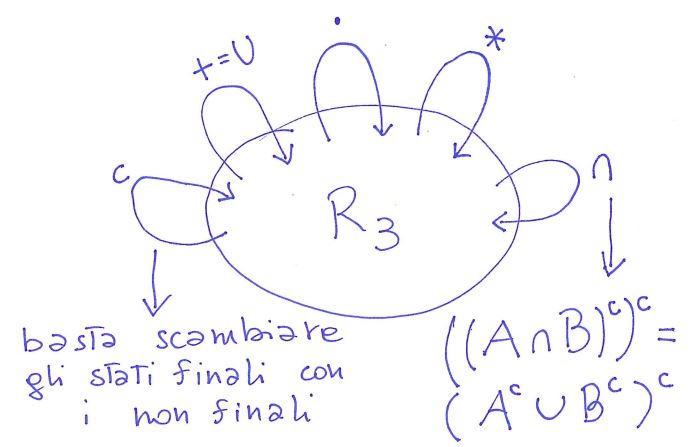
(Qualsiasi potenza del vuoto, eccetto la potenza 0, mi da il vuoto. Il vuoto alla zero, è un linguaggio alla zero, che da sempre la parola vuota, quindi l’unione di tutte le potenze mi da )

Adesso prendiamo la prima equazione

Dove e

Chiusura dei linguaggi regolari

Data la classe dei linguaggi regolari, quali sono le operazioni chiuse verso di esse?



Rispetto al complemento otteniamo sempre linguaggi regolari perchè se abbiamo un automa a stati finiti per un linguaggio regolare, l’automa complemento avrà stati finali e non finali invertiti, ciò non cambia lo status del linguaggio complementato .

L’intersezione è chiusa sui linguaggi regolari dato che per la legge di De Morgan diventa il quale è regolare dato che sono solo operazioni chiuse su linguaggi regolari.

Lezione del 19 Maggio 2022

Linguaggi liberi dal contesto e grammatiche di tipo 2

Problema dell’ambiguità

Esempio in italiano:

“Il prof. dice lo studente è un asino.”

Se letta in maniera diversa possiamo intuire che il professore sia un asino (dipende tutto dalla punteggiatura e dalla pronuncia).

Se viene letta senza entrambe non si capisce chi dei due sia un asino.

Definizione di ambigua:

Una grammatica è ambigua quando genera una parola ambigua, cioè quando la parola ammette due alberi di derivazione diversi.

Nota: l’albero di derivazione dà significato alla parola (il concetto sta in un albero che descrive la derivazione della parola, sulle foglie abbiamo la parola, i nodi sono i metasimboli, i sottoalberi sono definiti da regole di produzione e come radice abbiamo l’assioma).

Esempio di ambigua:

Grammatica per un linguaggio di programmazione “giocattolo” (atti alla didattica).

Simbolo iniziale di programma che produce input, comando e output. In questi programmi abbiamo solo la variabile .

I comandi possibili sono 3:

1. di assegnamento ;
2. comando senza , quindi esegue un comando verificando la condizione, nel caso sia falsa non accade nulla ;
3. comando con due comandi a seconda dei casi possibili.

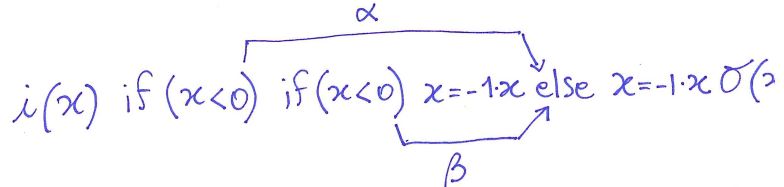
Abbiamo la sostituzione , quindi viene cambiato il segno alla variabile e viene assegnata ad essa.

In questo comando ha la funzione di “Test”

In quest’altro caso abbiamo due comandi a seconda dei casi.

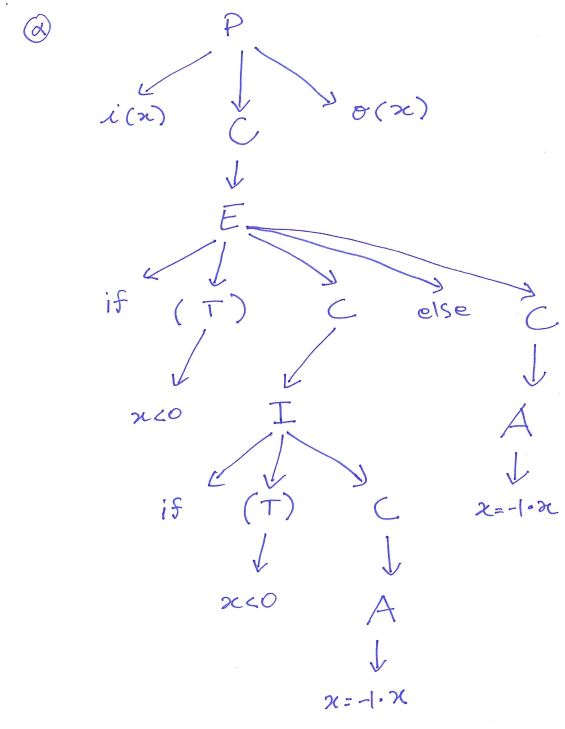
Unico test possibile

genera la seguente parola: (programmino)

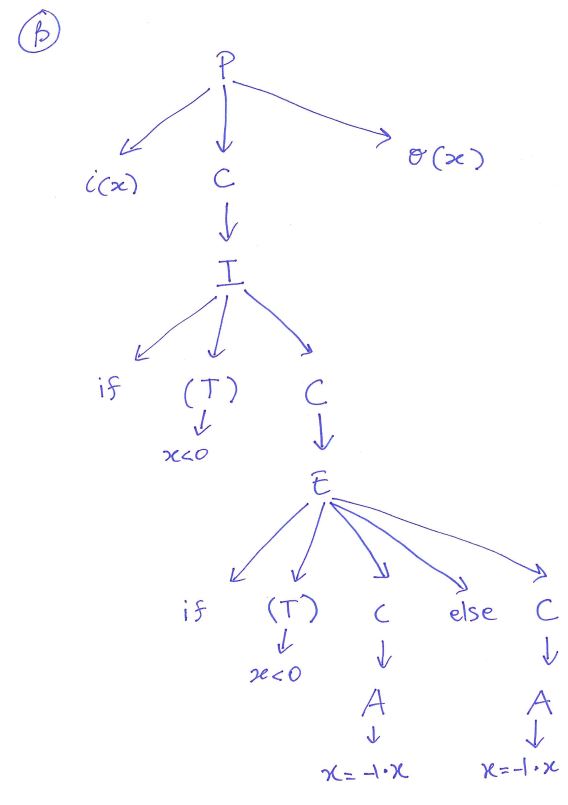


La quale ammette due alberi di derivazione per due programmi differenti e .

Programma



Programma

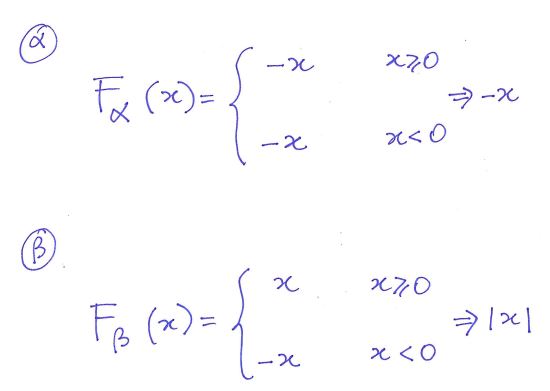


Il primo programma ha come primo comando un mentre nel secondo il primo comando è (Abbiamo un'inversione tra ed ).

Proviamo ad inserire come input una positiva e una negativa a entrambi i programmi:

* Caso :
  + positiva, eseguo il secondo ramo del primo comando, quindi diventa negativa ;
  + negativa, eseguo il primo ramo del primo comando, viene eseguito il test (di nuovo) e viene eseguito di nuovo il comando per rendere negativa ;
* Caso :
  + positiva, entriamo nel primo controllo del primo comando, dato che il test non va a buon fine il programma da in output senza cambiamenti;
  + negativa, entriamo nel primo controllo, dato che viene verificato entriamo nel secondo controllo, il quale trasforma la e la fa diventare negativa.

L’albero dà significato alla parola, in questo caso associa la funzione calcolata dal programma.



In conclusione il programma è ambiguo e è ambigua.

Definizione

Una grammatica è non ambigua se ogni parola generata non è ambigua.

Osservazione

A volte è possibile disambiguare una grammatica.

Nel nostro caso possiamo usare una convenzione:

“L’else è associato all’if più vicino”.

A volte no!

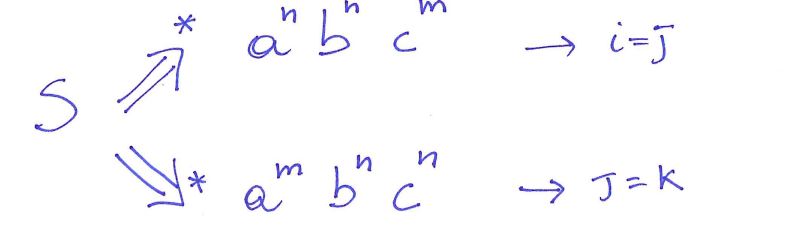
Definizione

Un linguaggio si dice “Inerentemente ambiguo” quando ogni che lo genera è ambigua.

Esempio:

Nota: è di tipo 2

Idea:



ammetterà due alberi di derivazione diversi.

Non c'è modo di trovare una grammatica per generare una grammatica con regole di produzione diverse e non ambigua.

Domanda:

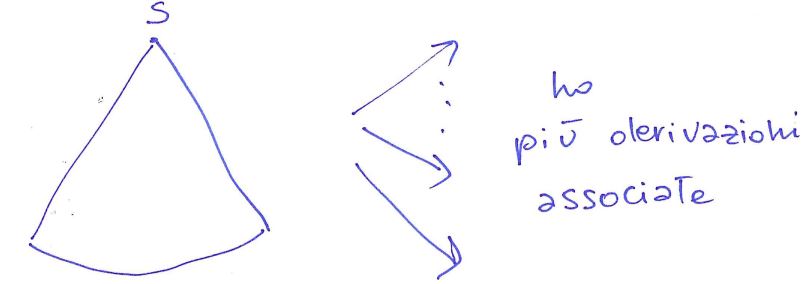
Un linguaggio regolare può essere inerentemente ambiguo?

Risposta:

NO

Dimostrazione

Ad ogni regolare corrisponde un . Da tale si ricava una di tipo 3 che è non ambigua.



Derivazioni leftmost

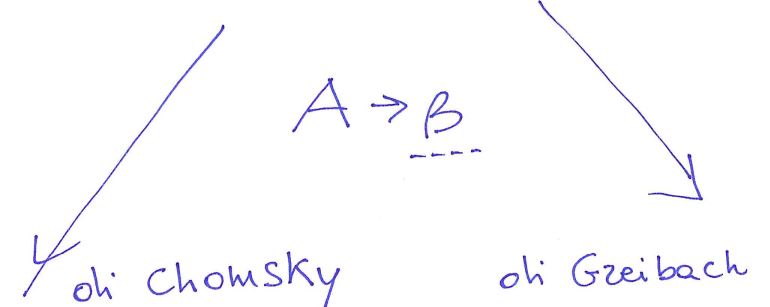
Dato un albero di derivazioni (con più derivazioni associate), attraverso questo metodo di derivazione viene espanso per primo il metasimbolo più a sinistra.

Esempio

Espandiamo prima poi e infine (se contiene altri metasimboli espandiamo quello più a sinistra al proprio interno).

Ad ogni albero di derivazione è associata un'unica derivazione leftmost, quindi si può in qualche modo eludere l’ambiguità.

Forme normali per di tipo 2 (senza )



A seconda di come è fatto variano tra le due classificazioni

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

La forma normale di Chomsky serve per dimostrare il Pumping Lemma (ultima lezione) mentre quella di Greibach serve per dimostrare l’equivalenza degli automi a pila (prossima lezione).

Esempio

dove

Ricaviamo FNC e FNG

FNG:

ok

ok

NO!!!

ok

ok

ok

FNC:

ok

ok

NO!!!

ok

ok

ok

NO!!!  
 ok

ok

NO!!!

ok

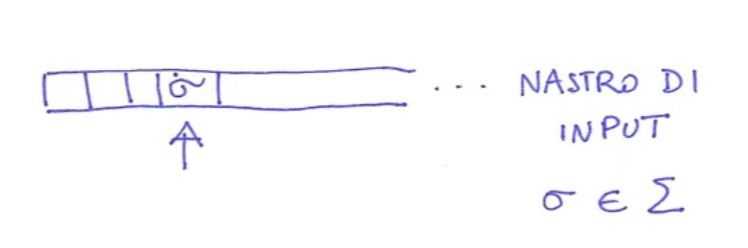
ok

Lezione del 24 Maggio 2022

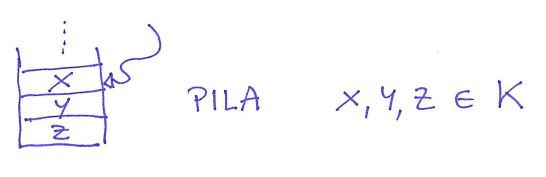
Riconoscitori a pila

Automi a pila per i linguaggi di tipo 2.

Sono un dispositivo composto da 2 parti hardware, nella prima abbiamo la parola da controllare inserita in un nastro di input simbolo per simbolo. Questo nastro è suddiviso in celle con in ognuna un simbolo appartenente all’alfabeto di input . Su questo nastro abbiamo anche una testina di lettura che scandisce la parola e si sposta tra i simboli.



La seconda parte di hardware è costituita da una memoria, indicata da una pila (dispositivo di memoria dove gli elementi vengono messi uno sopra l’altro). Nella memoria abbiamo un altro alfabeto . L’elemento più in basso è e quello più in alto è . Quando si legge nella pila, si parte da quello più in alto (in questo caso ).

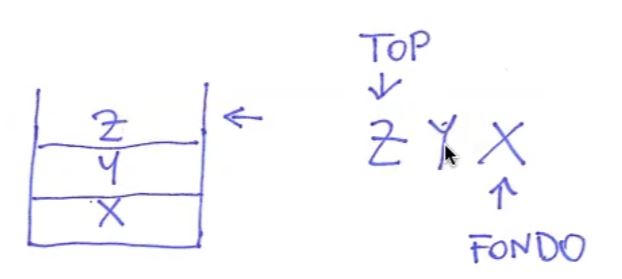


Definizione di Pila

Memoria ad accesso limitato (con una serie limitata di elementi dell’alfabeto ), con politica di accesso: LIFO (last in first out). L’ultimo elemento inserito è il primo da leggere e da rimuovere dalla pila. Si discosta dalla politica FILO (first in last out) per le memorie a coda.

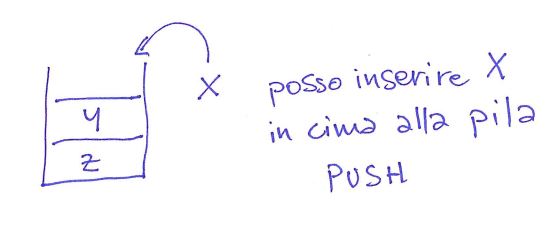
Notazione:

Disegnare la pila se possibile e descriverne il contenuto.

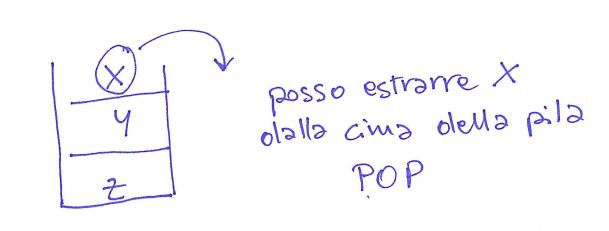


Operazioni di modifica della pila:

PUSH



POP



Formalizzo:

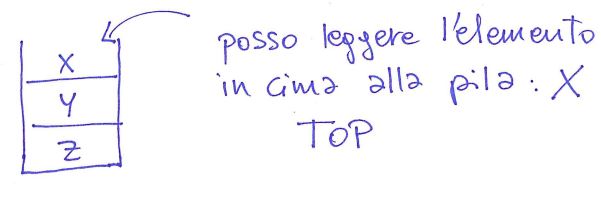
L'immagine di questa funzione è la pila modificata con a capo l’elemento rimosso. Come variabili abbiamo la pila e l’elemento più in alto.

se mentre se

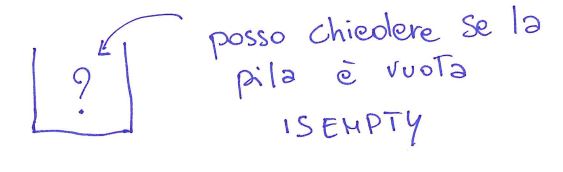
Come variabili abbiamo solo la pila. L’immagine è ciò che rimane dalla pila dopo l’estrazione, è se nella pila avevamo il carattere seguito da una serie di caratteri , mentre l’operazione non va a buon fine se la pila era vuota.

Operazioni di interrogazione della pila:

TOP



ISEMPTY



Formalizzo:

se mentre se

Prende come variabile la pila e restituisce l’elemento più in alto. Se la pila è vuota non va a buon fine.

se mentre se

Prende in input una pila e restituisce 1 se essa è vuota, mentre restituisce 0 se non lo è.

Definizione di riconoscitore a pila

(Usiamo i riconoscitori invece che gli automi a pila perchè sono più semplici e più funzionali per il riconoscimento di grammatiche di tipo 2)

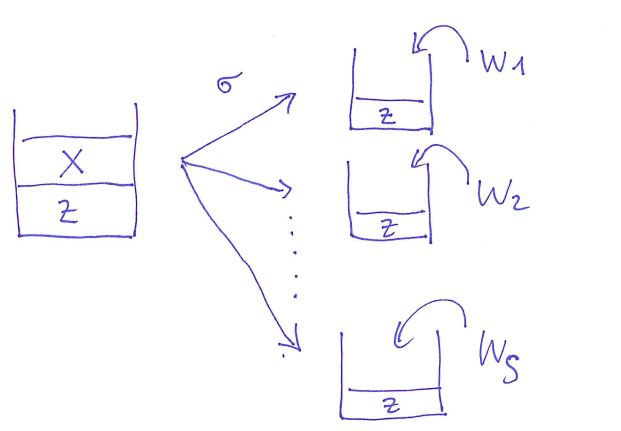
Un riconoscitore a pila è una tupla:

dove:

* è l'alfabeto di input ;
* è l’alfabeto della pila ();
* è il simbolo iniziale della pila ;
* funzione di evoluzione della pila (parole con elementi di ) con la scrittura:
  + con

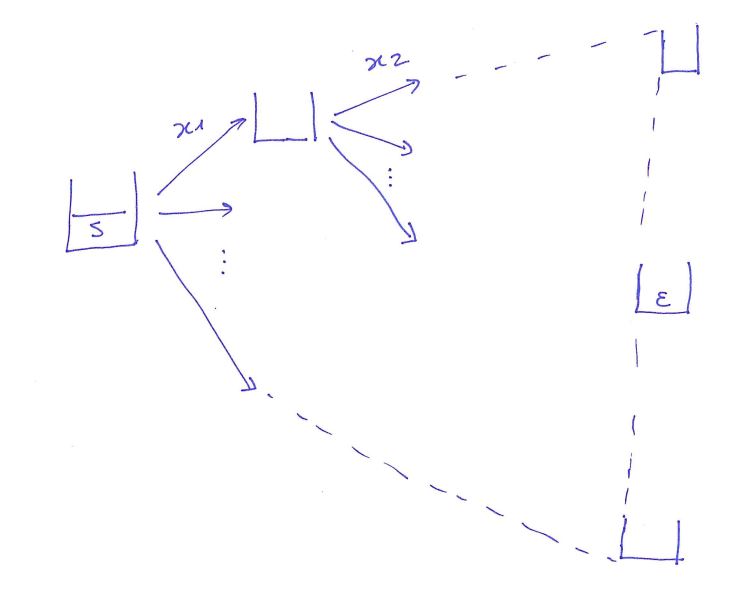
Con questa scrittura si indica che:

* è letto in cima alla pila (TOP) ;
* è letto sul nastro di input ;
* viene cancellato dalla pila (POP) ;
* viene scelto un NON DETERMINISTICAMENTE da inserire nella pila (PUSH);



Grafo di computazione dovuto a

Input



Abbiamo la pila iniziale, una volta inseriti man mano i simboli della parola del nastro la pila viene modificata in tutti i modi possibili secondo la funzione . Possiamo considerare le pile secondo i “livelli” , cioè il numero di simboli inseriti. Il modo in cui cambiano le pile è attraverso una scelta non deterministica.

La computazione secondo la parola è una delle pile del livello finale.

La parola è accettata oppure no?

Criterio di accettazione

La parola si dice accettata se nel grafo di computazione di esiste un cammino che partendo dalla pila mi porta ad una pila vuota (). Controlleremo tramite ISEMPTY.

Formalizzo:

Configurazione

(parte di input ancora da leggere)

Esempi

configurazione iniziale

configurazione finale accettante (finale perchè ho al posto di e accettante perchè ho al posto di )

Passo di computazione

Configurazione configurazione successiva

Scriveremo:

Linguaggio riconosciuto da

dove

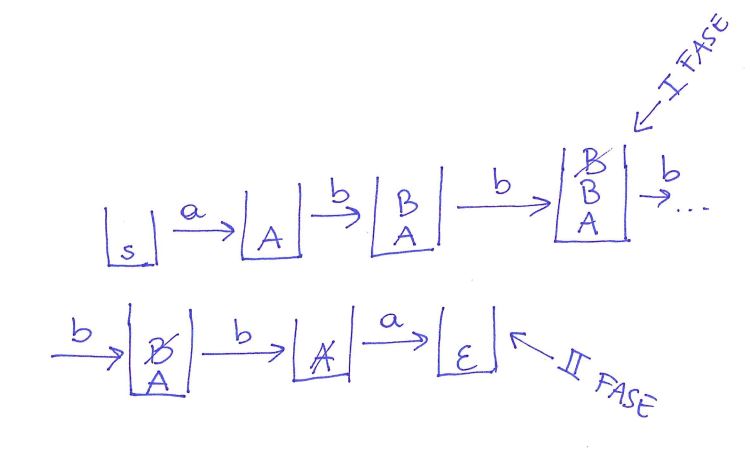
è: zero o più passi di derivazione

Esempio:

Linguaggio: palindrome

es:

Idea:



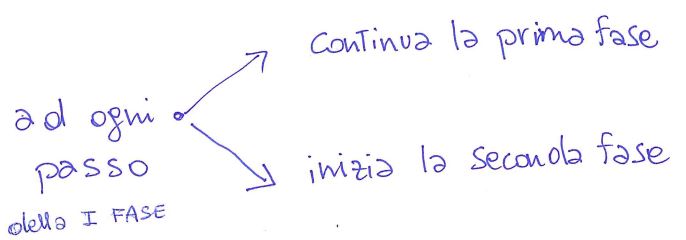
Quando raggiungo la metà della parola, confronto la seconda con i simboli nella pila cancellandoli man mano.

Problema:

Come riconoscere il centro della parola in input?

Risposta:

Si usa il non determinismo



Cima della pila

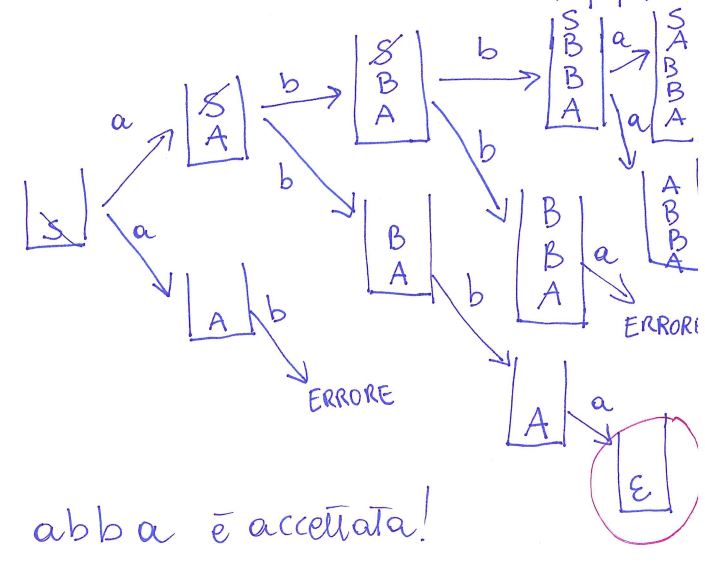
siamo nella prima fase, ovvero inserimento dei simboli nella pila

siamo nella seconda fase ovvero confronto dei simboli di input con il contenuto della pila.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  | - |
|  | - |  |

Dove - sta per “errore”

Grafo di computazione di .



Lezione del 26 Maggio 2022

Domanda:

Qual’è la classe dei linguaggi riconosciuti dai riconoscitori a pila?

Risposta:

I linguaggi di tipo 2.

Attenzione:

Il modello dei riconoscitori a pila deve essere NON DETERMINISTICO:

e non |

Infatti:

Teorema

è generato da di tipo 2 è riconosciuto da un riconoscitore a pila.

Lo stesso vale per gli automi a pila, con un riconoscitore a stati che muove la testina di input cambiando ogni volta lo stato corrente, alla fine il riconoscitore può decidere di accettare o a pila vuota o controllando se lo stato raggiunto sia finale.

La dimostrazione di questo teorema è costruttiva in entrambi i sensi.

Dimostrazione 1

Da riconoscitore a pila a di tipo 2

Sia posso costruire di tipo 2:

contiene la regola:

Esempio:

su

dove

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  | - |
|  | - |  |

è l’assioma

La ottenuta è la forma normale di Greibach di:

, , ,

Dimostrazione 2

Da di tipo 2 ad riconoscitori a pila.

Avendo una di tipo 2 dobbiamo per prima cosa trasformarla in forma normale di Greibach:

.

Costruisco

così definita:

Esempio

per : ,

Trasformo in di Greibach:

diventa:

e

diventa:

con la regola già esistente

Definisco

dove

|  |  |  |
| --- | --- | --- |
|  |  | - |
|  | - |  |

Attenzione: non è il miglior riconoscitore per (dato che quello che abbiamo trovato è non deterministico), infatti ne esiste uno deterministico:

dove

|  |  |  |
| --- | --- | --- |
|  |  | - |
|  |  |  |
|  | - |  |

Note:

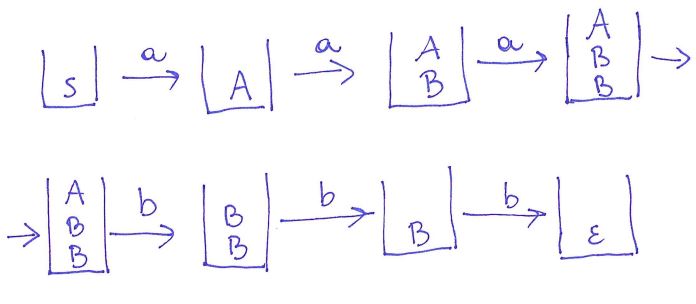
in cima:

sono nella prima metà e inserisco in cima una

in cima:

sono nella seconda metà e cancello le dalla pila

Esempio:

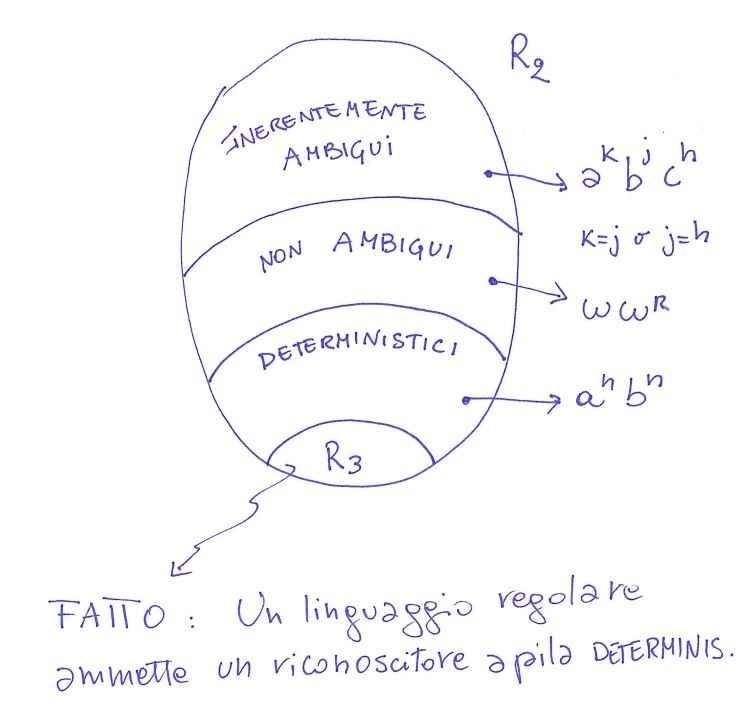


Osservazione:

senza il simbolo , verrebbe accettata!

ma non nel formato

Riassumendo:



Dimostrazione, idea

esiste simulo usando la pila: metto lo stato nella pila\*

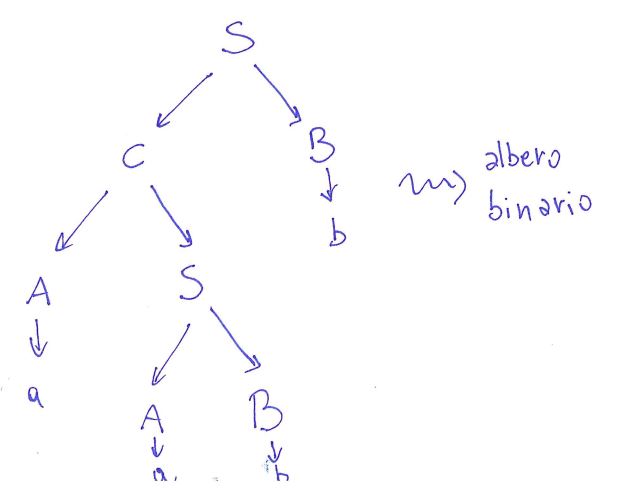
\*”ISEMPTY” viene sostituito da “ISFINALSTATE”!

Alberi binari

Forma normale di Chomsky

si hanno alberi di derivazione binari

es:

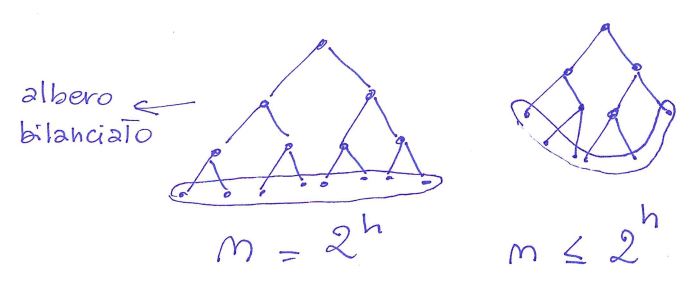


Albero di derivazione per

Relazione:

numero di foglie

altezza



In generale ho:

da cui:

Questa relazione ci servirà per dimostrare la correttezza del PUMPING LEMMA

Lezione del 31 Maggio 2022

Pumping Lemma

Strumento utile per dimostrare che i linguaggi non sono di una determinata grammatica. Ne abbiamo uno per linguaggi regolari ed uno per i linguaggi liberi da contesto. Vedremo solo quelli per i linguaggi di tipo 2.

Esprime una condizione necessaria (non sufficiente, quindi non possiamo dire che sia di tipo 2 ma possiamo verificare che non lo sia) per i linguaggi di tipo 2.

Nota:

* non soddisfa il lemma non è di tipo 2 ;
* soddisfa il lemma può essere di tipo 2 (non certo).

Enunciato

Per ogni di tipo 2 esiste una costante tale che:

per ogni con esiste una scomposizione in

(scompone la parola di una certa lunghezza del linguaggio in 5 fattori con fattore centrale)

che soddisfa:

1. (se prendo il secondo ed il penultimo fattore e li metto vicini essi devono avere lunghezza , quindi almeno uno dei due è un simbolo) ;
2. (la lunghezza della parte centrale della scomposizione deve essere minore o uguale ad ) ;
3. (abbiamo tante parole simili nella seguente forma nel linguaggio) .

Dimostrazione

Per dimostrarlo prendiamo ogni frase e dettaglio dell’enunciato e vediamo le implicazioni.

* “Per di tipo 2 esiste ” ;

Se è di tipo 2 allora ammette una in forma normalizzata di Chomsky. Fisso quindi dove numero di variabili in .

* “per ogni tale che ” ;

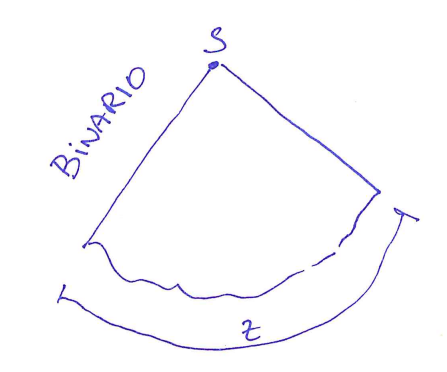
Dato che è di tipo 2, allora viene creato da un albero binario e quindi segue le regole descritte nella lezione precedente:

numero di foglie

altezza

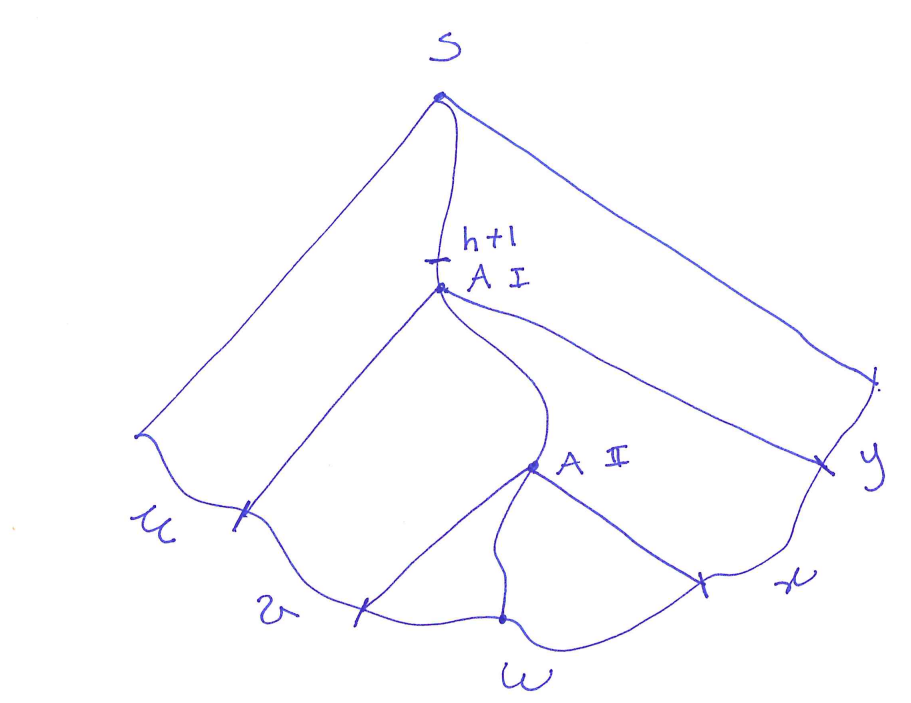
da cui:

Quindi:



* “esiste una scomposizione di in ” ;

si ricava dall’albero di derivazione per :



Sui rami non trovo più ma direttamente la scomposizione in 5 fattori.

* considero
  + il ramo più lungo dell’albero ;
* risalgo
  + dal fondo di nodi (etichettati da variabili, ma dato che le variabili sono e risalgo fino al nodo esimo troverò almeno una ripetizione) ;
* per il principio della piccionaia due nodi sono etichettati con la stessa variabile che chiameremo: ;
* la scomposizione si ricava da questi due nodi, la prima e la seconda occorrenza di () , staccando il primo sottoalbero con radice ottengo sulle foglie mentre staccando il secondo sottoalbero con radice ottengo un sottoalbero con foglie ed infine i simboli che compongono sono generati da .
* “la scomposizione soddisfa 1) 2) 3)” :

Condizione 1

e non possono essere contemporaneamente perchè e sono distinti (vero per come abbiamo composto l’albero di derivazione in modo da rendere quelle due variabili non sovrapponibili e distinte).

Condizione 2

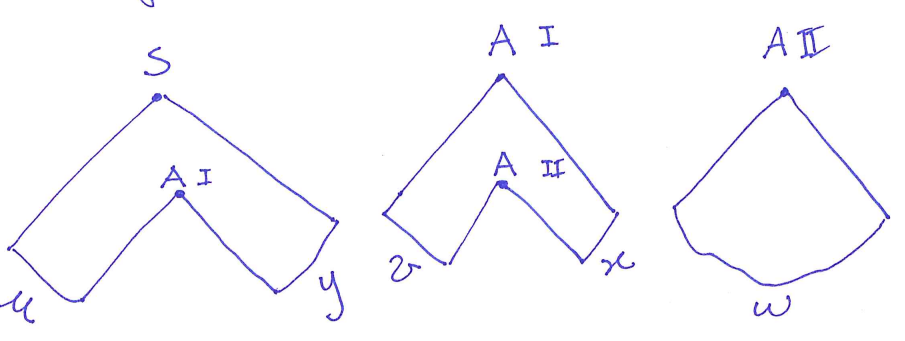
si ha:

( altezza del sottoalbero)

Verificata.

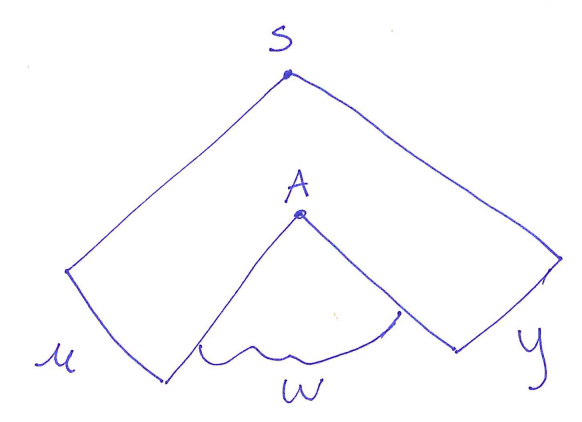
Condizione 3

Ritaglio dell’albero



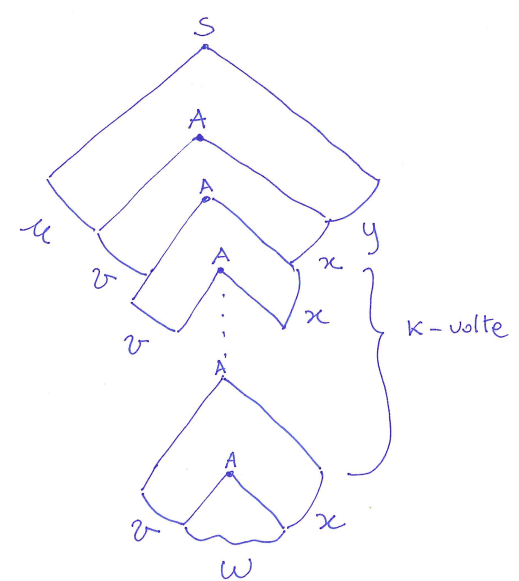
Caso :

devo mostrare che



Caso

devo mostrare che ( e si ripetono volte)



Applicazione del pumping lemma

Fatto:

non è di tipo 2

Dimostrazione

Considero la parola

, e faccio vedere che qualunque sua scomposizione in non soddisfa contemporaneamente 1) , 2) , 3) .

Prendiamo come

Per si hanno i seguenti casi:

* contiene sia ;
* contiene o (non consideriamo la seconda opzione per simmetria) ;
* contiene o o (non consideriamo la seconda e la terza opzione per simmetria) ;

Studiamo i casi:

Primo caso

quindi

La condizione 2 è falsa

Secondo e terzo caso

In entrambi questi casi sia che contenga , sia che contenga esso non contiene le

Considero:

(Cioè che la condizione 3 sia vera)

allora (dato che il numero di è ) (dato che )

da cui si ottiene che non vale la condizione 1 dato che in questo caso per far si che si verifichi la condizione 3 e la formula espressa precedentemente.