Paradigmi e linguaggi di programmazione

Linguaggi formali e Modelli per l'Informatica

Sommario

- Modelli per l'Informatica
  - Introduzione
- Linguaggi formali
  - Definizione
  - Espressioni regolari
  - Grammatiche generative
    - E Classificazione di Chomsky
  - Traduzione come comprensione di un linguaggio
- Introduzione al Natural Language Processing
- Il formalismo delle macchine (gli automi)
  - E tesi di Church risoluzione meccanica di problemi
  - I compilatori

Richiami linguaggi formali

Bibliografia

- Dino Mandrioli, Paola Spoletini, Informatica teorica, Editore Città studi Milano- collana: Informatica, 2011
- Dino Mandrioli, Carlo Ghezzi, Informatica teorica, Editore Città studi Milano - collana: Informatica, 1989

- Han, Zdravko Dovedan, Kristina Kocijan, and Vjera Lopina. "PYTHON as Pseudo Language for Formal Language Theory." MIPRO 2016-Computers in education (CE). 2016.
  - http://docs.mipro-proceedings.com/ce/ce_33_3859.pdf
  - https://github.com/dragonwasrobot/formal-language

C'è ancora ricerca …

EUDroid: a **formal language** specifying the behaviour of IoT devices
P Buono, F Cassano, A Legretto, A Piccinno - IET Software, **2018** - IET
Recent technologies are offering today many possibilities to end users, which ask for continuous support in a variety of situations. Internet of things (IoTs) and the proliferation of smart devices are offering many opportunities that raise the need to standardise protocols …
☆ 99 Citato da 7 Articoli correlati Tutte e 2 le versioni Web of Science: 3 »

Kevm: A complete **formal** semantics of the ethereum virtual machine
E Hildenbrandt, M Saxena, N Rodrigues… - **2018** IEEE 31st …, **2018** - ieeexplore.ieee.org
… **2018**, Everett Hildenbrandt … is a rewriting based framework for defining executable semantic specifications of programming languages, type systems and **formal** analysis tools … Given the syntax and semantics of a **language**, K generates a parser, an interpreter, as well as **formal** …
☆ 99 Citato da 87 Articoli correlati Tutte e 5 le versioni »

Linguaggi naturali, formali e di programmazione

Linguaggio è un insieme generalmente infinito di stringhe caratterizzate da qualche particolare proprietà

- **Naturali** (nati spontaneamente):
  - non rigorosamente definiti, in continua evoluzione
  - spesso presentano delle ambiguità;
  - hanno però una enorme potenza espressiva.
- **I linguaggi formali**:
  - completamenti definiti mediante regole esplicite,
    - sempre possibile determinare la correttezza (grammaticale) di una proposizione;
  - il significato di ogni frase è sempre privo di ambiguità.
  - però hanno un potere espressivo limitato.

Linguaggi naturali, formali e di programmazione

- **Cos'è il Linguaggio Naturale ?**
  - Strumento di comunicazione tra persone;
    - Fatti, idee e conoscenze sul mondo esterno ed interiore
    - Emozioni
    - Ordini
- **Cos'è un Linguaggio Formale ?**

Dato un insieme di simboli Σ detto alfabeto, un linguaggio formale è un sottoinsieme di tutte le possibili concatenazioni dei simboli:

$$L \subseteq \Sigma^*$$

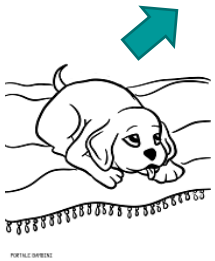
Un linguaggio formale non è ambiguo (una concatenazione di simboli ha una interpretazione univoca) ed esprime le sue regole in maniera canonica

Da dove derivano?

Linguaggi naturali

- All’origine dei linguaggi naturali vi è la necessità di comunicare concetti in modo comprensibile .

- Il primo passo è attribuire un significato (*semantica*) ad una forma, sia essa un disegno, un suono o una o più parole che rispettano alcune regole (*sintassi*).



Cucciolo di cane

Linguaggi naturali

- Il linguaggio naturale è spesso ambiguo:
  - *Rapina in banca con rivoltella da centomila euro* (rapina effettuata con una rivoltella costosa, o rapina da centomila euro?)
  - *Ci scusiamo dei possibili fastidi causati porgendo cordiali saluti* (fastidi causati dai saluti o si porgono cordiali saluti a seguito dei fastidi?)
  - *Mario ha una vecchia credenza* (in senso di pregiudizio o un mobile)
  - *Mario è un fulmine (*chi è Mario?)
- Per superare l'ambiguità della frase occorre il <u>contesto:</u>
  - ovvero la situazione particolare in cui le frasi vengono usate, costituita dal resto della conversazione, dall'ambiente fisico in cui avviene, dall'identità degli interlocutori

Linguaggi naturali

- Il linguaggio naturale è spesso ridondante:
  - la **ridondanza** è l'uso di parole la cui omissione non costituisce una sostanziale perdita di significato.
  - lunghe perifrasi per un semplice concetto:
  - *…. Mi sia consentito di esprimere in questa sede e davanti a tale consesso di professori il mio sia pur inesperto pensiero..*
  - *Traduzione "io penso"*
- Il linguaggio naturale può avere frasi autoreferenziali:
  - L'autoreferenza è un fenomeno nel linguaggio naturale o formale consistente in una frase o formula che si riferisce a se stessa
  - *La proposizione «Questa frase è scritta in italiano» è autoreferenziale*
  - *La proposizione «Questa frase è falsa» è vera o falsa?*
  - In informatica l'autoreferenza viene riscontrata nel concetto di ricorsione, in cui un algoritmo viene espresso in termini di se *stesso.*

I linguaggi formali

- Si possono costruire eliminando:
  - Ambiguità
  - Ridondanza
  - Autoreferenza
- ovvero si impedisce che a un'informazione si attribuiscano più significati.
- Il linguaggio formale può essere utilizzato in diversi contesti:
  - scientifico per indicare una notazione o un formalismo con sintassi e semantica definite in modo preciso
  - legale, amministrativo, in cui si presta cura all'uso corretto di una particolare terminologia al fine di eliminare o ridurre le ambiguità di interpretazione.
  - artistici, per esempio nella composizione della musica.

Linguaggi formali vs naturali

- Lo studio della semantica delle lingue naturali è ***descrittivo*** e a posteriori
  - I segni linguistici si creano ed evolvono informalmente nei processi di comunicazione
  - in semiotica, un segno è definito come «qualcosa che sta per qualcos'altro» ed è considerato una unità discreta di significato
  - I *linguisti* osservano e catalogano gli usi dei segni dove le relazioni lessicali (sinonimia, iponimia) sono sfumati
- La semantica di un linguaggio formale invece è ***prescrittiva*** e a priori
  - I significati sono assegnati convenzionalmente e codificati in regole che sono imposte nei processi comunicativi
  - Le regole devono essere coerenti e le assegnazioni precise
  - I *logici* definiscono e realizzano questi segni e regole

Linguaggi naturali, formali e di programmazione

Un elaboratore può riconoscere e generare solo Linguaggi Formali, attraverso l'utilizzo di modelli e algoritmi

- *Linguaggi di programmazione*
  - definiti come *il mezzo che ci permette di comunicare al computer la sequenza di operazioni da effettuare per raggiungere un obiettivo prefissato*.

Cosa serve per analizzare un linguaggio?

- **CONOSCENZA LINGUISTICA**
  - tutta la conoscenza che ha a che vedere con il linguaggio: cos'è una parola? Quali sono le regole per costruire una frase? Qual è il significato di un sintagma?
- **MODELLI (teorie)**
  - i modelli linguistici hanno lo scopo di catturare la conoscenza linguistica e rappresentarla in una forma comprensibile per il computer
- **ALGORITMI**
  - strumenti per manipolare i modelli e le strutture linguistiche necessarie per l'analisi e la comprensione del linguaggio

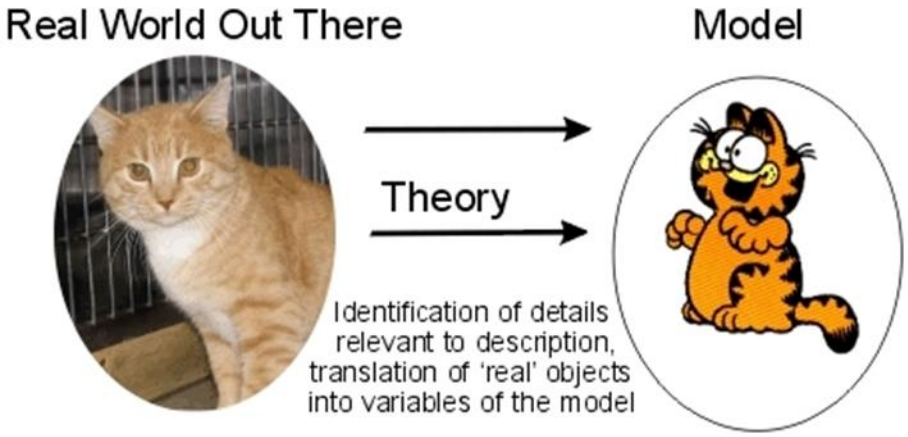
Sintagma (ingl. ***phrase***): unità sintattica significativa autonoma; nella frase «Pietro è affezionato a Paolo», i sintagmi sono tre: soggetto (Pietro), predicato (è affezionato), complemento di termine (a Paolo).

Modelli & Ingegneria

Ogni fase di progetto si fonda sull'uso di modelli, in quanto risulta spesso impossibile o poco pratico verificare se la soluzione di un problema sia adeguata o meno, applicandola direttamente al mondo reale.

Distinguiamo

- ***Modelli fisici***
  - situazione ridotta di quella reale
- ***Modelli formali***
  - opera su oggetti matematici che rappresentano le astrazioni delle entità reali che devono essere modellate



Modelli formali

- **MODELLI PROCEDURALI:**
  - Automi a Stati Finiti
  - Trasduttori a Stati Finiti
  - Markov Models … …
- **MODELLI DICHIARATIVI:**
  - Grammatiche regolari
  - Context Free Grammar …
- **MODELLI LOGICI:**
  - Calcolo dei Predicati
  - Logica Descrittiva…

➢Solitamente un modello procedurale ha una sua controparte in un modello dichiarativo

=>ad es. *automi vs. grammatiche regolari*

➢Un modello può essere più o meno complesso da un punto di vista computazionale

=>ad es. *le Context Free Grammar sono più complesse di quelle Regolari*

➢Nei diversi modelli possono generalmente essere integrati elementi di probabilità (*modelli probabilistici*)

Modelli formali

I ***modelli formali*** richiedono

- *Formalizzazione del problema*:
  - la traduzione del problema reale in una notazione propria di qualche formalismo matematico
- *Risoluzione del problema formalizzato*
  - mediante gli strumenti offerti dal formalismo scelto
- *Interpretazione dei risultati ottenuti dal modello*
  - allo scopo di dedurre o valutare le scelte di progetto

Modelli formali

Programmi

- Modelli formali per l'informatica

→ devono *conformarsi* alla sintassi e semantica del linguaggio di programmazione scelto

La situazione

Realizzazione di funzionalità S/W mediante lo schema:

- elaboratore programmabile, fornisce azioni elementari
- programmazione dell'elaboratore tramite un algoritmo
  - specifica una sequenza finita di azioni elementari
- si fornisce un input, si riceve un output

Modelli formali

I modelli a cui si fa riferimento

- ***Automi***
  - ASF (a stati finiti), PDA (a pila), TM (macchina di Turing)
  - Diverse azioni elementari fornite => diverso stile computazionale

- ***Grammatiche***
  - esprimono la rappresentazione di dati e programmi
  - definiscono la “sintassi” del linguaggio di programmazione

Modelli formali

- *Qual è lo stile di programmazione promosso dagli automi?*
  - input e output
    - sequenze di un alfabeto ristretto
  - azioni elementari
    - cambio stato, accesso ad un pila, scrittura su nastro

- elaboratore
  - automa astratto
- il listato del programma
  - una specifica di insiemi matematici e funzioni:
    - insieme degli stati, degli ingressi, delle uscite,...
    - funzione di transizione dello stato (dinamica del sistema)

Modelli formali

Automi

- *Quale ausilio all'ingegneria?*
  - grande espressività concettuale
  - pragmaticamente di difficile utilizzo

Modelli formali

Automi

- La programmazione di un automa è di difficile comprensione
  - difficile capire se tutta la casistica è stata considerata
  - difficile capire se il diagramma degli stati è corretto
  - difficile intuire il funzionamento "ad occhio"
  - difficile manutenzione
- Non si presta alla soluzione di problemi fondamentali:
  - operazioni matematiche su interi, reali
  - operazioni su strutture dati complesse
    - (es.: elenco dipendenti)
- Anche nei casi più semplici, si hanno automi molto complessi

Modelli formali

Programmazione e Astrazione

- Come in generale accade nei modelli scientifici è necessario avere il giusto livello di astrazione

- invece di costruire un automa pragmaticamente più espressivo:
  - si definisce un ***linguaggio di programmazione***
  - specificando una grammatica
    - ***sintassi del linguaggio***
  - descrivendo l'effetto dell'esecuzione di un programma
    - ***semantica del linguaggio***

Modelli formali

- ***lo scopo***
  - fornire un alto livello d'astrazione sulla computazione
  - mostrare un elaboratore (astratto) che può essere espressivo, potente e utile per le applicazioni

Linguaggio di Programmazione

- definisce un elaboratore astratto di alto livello
- con azioni elementare più potenti
- con più flessibilità nell'aggregare tali azioni
- semplificando la specifica di algoritmi

Modelli formali

Linguaggio di Programmazione

- è definito da due aspetti:
  - sintassi
    - quali programmi sono sintatticamente corretti?
    - es.: qual'è la grammatica formale del linguaggio?
  - semantica
    - quali programmi sono semanticamente corretti?
    - qual'è l'effetto dell'esecuzione del programma?

Sintassi vs. Semantica

- Programma sintatticamente corretto
  - soddisfa le regole sintattiche
  - cioè, è una sequenza di parole o simboli appartenenti alle giuste categorie sintattiche
- Programma semanticamente corretto
  - soddisfa le regole semantiche
  - dice se il programma ha senso, se sarà possibile eseguirlo
- Esempio:
  - "Il cane gioca in borsa" : è sintatticamente corretta, ma non semanticamente
  - 3+true=false è sintatticamente corretta, ma non semanticamente

Semantica

Distinguiamo due aspetti fondamentali

- correttezza semantica (typing)
  - quali tipi di dato possono essere elaborati?
  - quali operatori applicabili ad ogni dato?
  - quali regole per definire nuovi operatori?
- effetto dell'esecuzione (semantica operazionale)
  - qual è il singolo effetto di ogni azione elementare?
  - qual è l'effetto dell'aggregazione delle azioni?
  - e quindi:
    - qual è l'effetto dell'esecuzione di un certo programma?
    - qual è la funzione realizzata?

Linguaggi di programmazione e architettura

Se non esiste **IL** linguaggio di programmazione.

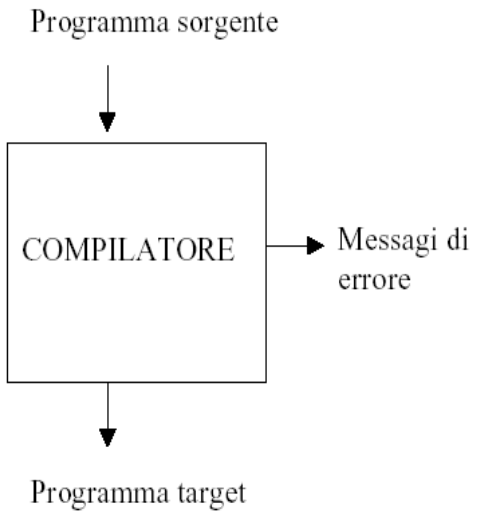
=> E' necessario disporre di *linguaggi diversi*:

- che consentano di rappresentare in modo espressivo e flessibile le computazioni di interesse

- e per ognuno di essi, e indipendentemente dall'H/W, di meccanismi per
  - trasformare l'algoritmo specificato da un programma
  - in una esecuzione nell'architettura H/W usata

Traduttore

- Cioè un programma che legge il programma sorgente, scritto in un determinato linguaggio e lo traduce in un altro programma equivalente
- Cerca una corrispondenza fra frasi appartenenti a due linguaggi diversi



Nel caso di linguaggi artificiali può essere intesa in modi differenti:

- Compilazione da un linguaggio ad alto livello al codice macchina
- Trasformazione di un documento HTML al formato Postscript

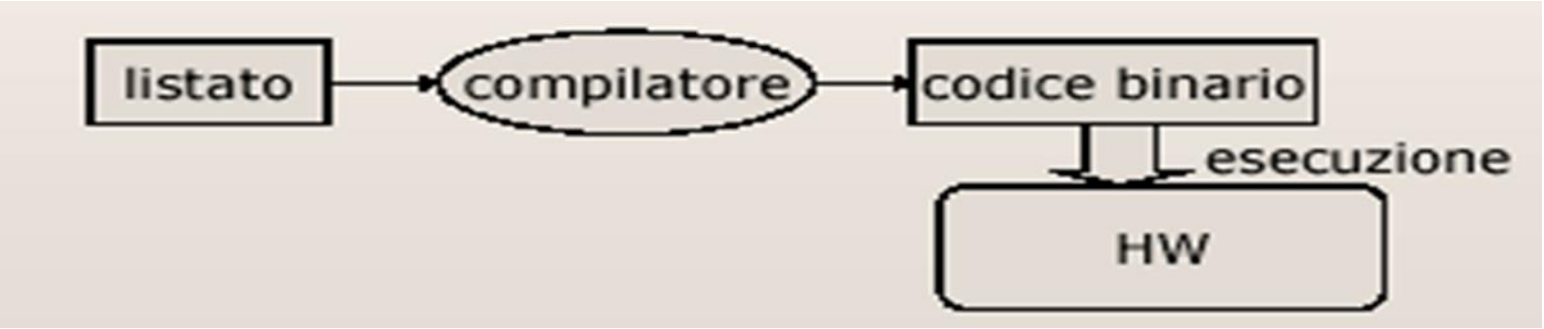
Traduttore

- Cerca una corrispondenza fra frasi appartenenti a due linguaggi diversi
- Si individuano due approcci basati sui due modelli per la programmazione
  - ***Riconoscitivo***: si avvale di automi che effettuano la traduzione voluta
  - ***Generativo***: impiega degli schemi sintattici di traduzione per generare le coppie di frasi che corrispondono nella traduzione

Traduttore

Schema a compilazione:

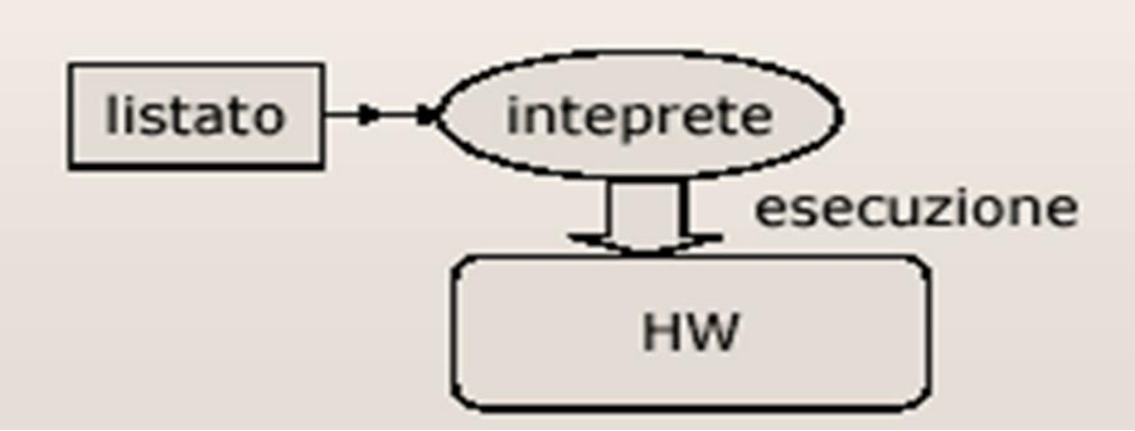
- Una applicazione S/W, chiamata compilatore
  - prende in ingresso il programma
    - listato, o codice sorgente
  - lo traduce nel programma per l'H/W specifico
    - codice binario (es.: un file ".EXE" in Windows)
    - direttamente eseguibile



Traduttore

Schema a interpretazione:

- Una applicazione S/W, chiamata interprete
  - esegue l'applicazione "al volo":
    - legge il listato, istruzione per istruzione
    - per ognuna, fa eseguire all' H/W le operazioni corrispondenti

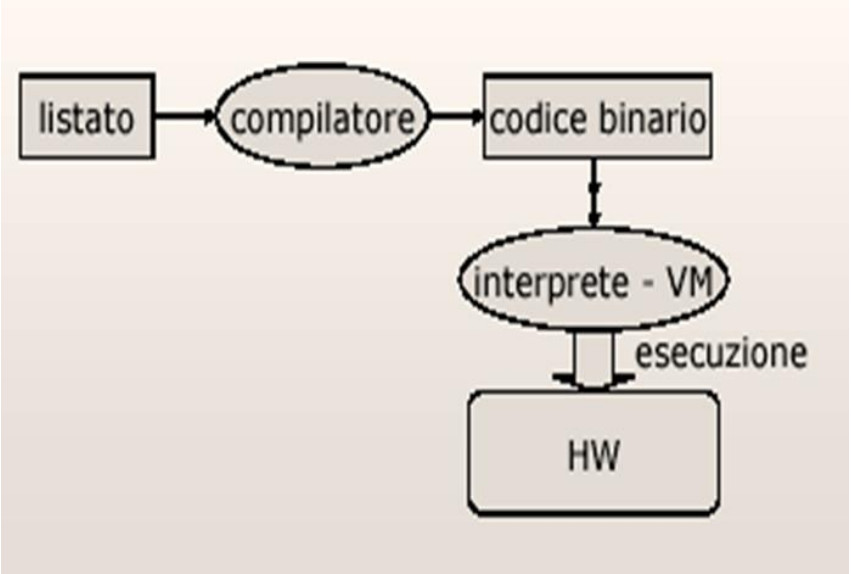


La Java Virtual Machine

Schema ibrido compilazione/interpretazione

Java Virtual Machine

- Il compilatore:
  - a partire dal listato in Java
  - crea del codice binario
    - non per una architettura specifica (Intel, Linux,...)
    - ma per una architettura virtuale (JVM)
- Interpretazione:
  - l'interpretazione viene effettuata dalla JVM
  - è un "simulatore" della architettura virtuale per le architetture effettive



Linguaggi formali

- **Linguaggi formali**
  - Linguaggi formali

Linguaggi formali

- *Perché studiare i linguaggi formali e la teoria degli automi?*
  - Perché stanno alla base dei compilatori correnti, delle espressioni regolari, dei parser, dei web-scrappers, del natural language processing (NLP), delle macchine a stato basate sulle catene di Markov, …

Linguaggi formali

- Linguaggio formale è un linguaggio in cui la forma delle frasi (***sintassi***) e il loro significato (***semantica***) sono definiti in modo preciso e algoritmico

- Linguaggio formale è un linguaggio per cui risulta possibile progettare una procedura algoritmica che verifichi la correttezza grammaticale delle frasi e ne calcoli il significato

➢ ***Definizione formale***

Un linguaggio formale è una struttura matematica, costruita a partire da un alfabeto mediante regole assiomatiche (grammatiche formali) o modelli matematici (automi o macchine di Turing)

Definizioni

Alfabeto A: un insieme finito e non vuoto di simboli atomici

Esempio: A={a,b}

Stringa o parola dell'alfabeto: è una sequenza finita di simboli dell'alfabeto con lunghezza pari al numero di caratteri

Esempio: a, b, ab, aba, bb, bbaaa,…

- ***01000010*** è una stringa definita sull'alfabeto binario {0, 1}. La sua lunghezza è 8. Essa appartiene quindi a {0, 1}.
- La sequenza infinita ***010101*** ... non appartiene a {0, 1}.
- ***abracadabra*** è una stringa definita sull'alfabeto italiano. La sua lunghezza è 11.

Definizioni

- Si definisce **concatenazione** l'operazione (denotata °) che consiste nel giustapporre due stringhe. La concatenazione è un'operazione associativa ma non commutativa.
  - salva ° gente = salvagente
  - abb ° bba = abbbba ≠bba ° abb = bbaabb
  - ((abb ° b) ° ba) = (abb ° (b ° ba)) = abbbba

- Per indicare la ripetizione di simboli o più in generale la concatenazione di due o più stringhe uguali si usa il simbolo di potenza.
  - ab^4a = abbbba
  - w^3= www
  - se x = cous, con x^2 si indica la stringa couscous.

Definizioni

Linguaggio Il linguaggio ***L*** su un alfabeto A è un insieme di stringhe di A*, sottoinsieme di tutte le stringhe di lunghezza arbitraria ma finite possibili su A. Esiste la stringa vuota.

- *Frase di un linguaggio*: stringa finita appartenente a quel linguaggio

Dato l'alfabeto {a, b}, l'insieme $L = \{a^n b^n \mid n \geq 0\}$
è il linguaggio di tutte le stringhe costituite da una sequenza di n a ($n \geq 0$), seguite da altrettante b. $aaabbb \in \Lambda$; $\varepsilon \in L$; $aaabb \notin \Lambda$.

Dato l'alfabeto {I, V, X, L, C, D, M}, l'insieme di tutti i numeri da 1 a 3000 rappresentati come numeri romani è un linguaggio.

Dato l'alfabeto {0, 1} l'insieme di tutte le stringhe che contengono un numero pari di 1 è un linguaggio.

Definizioni

Esempio ***linguaggio naturale***
il linguaggio delle frasi in italiano

- Alfabeto: A={a,b,...,z,<spazio>}
- Stringhe dell'alfabeto:
"casa", "la porta", "agsdh asg dh"
- Frasi della lingua italiana:
"il cane dorme",...

*Esempio: **linguaggio di programmazione***

alfabeto di un linguaggio di programmazione

- A={if, else, = A, 0,=,+,1,2,(,)}
- Stringhe:

A=a+2,if (A==0)A=A+2,A=A+A, if else A, if=A,…..

- Frasi

A =A+A, A=A+2, if (A==0) A=A+2

Definizioni

- Non tutti i linguaggi su un dato alfabeto sono interessanti.
- Si cerca un linguaggio le cui stringhe hanno struttura particolare:

- il linguaggio costituito da stringhe di parentesi bilanciate del tipo: (( )(( ))( ),
- il linguaggio costituito da espressioni aritmetiche contenenti identificatori di variabili e simboli delle quattro operazioni,
- il linguaggio costituito da tutti i programmi sintatticamente corretti (cioè accettati da un compilatore senza segnalazione di errore) scritti nel linguaggio C.

Definizioni

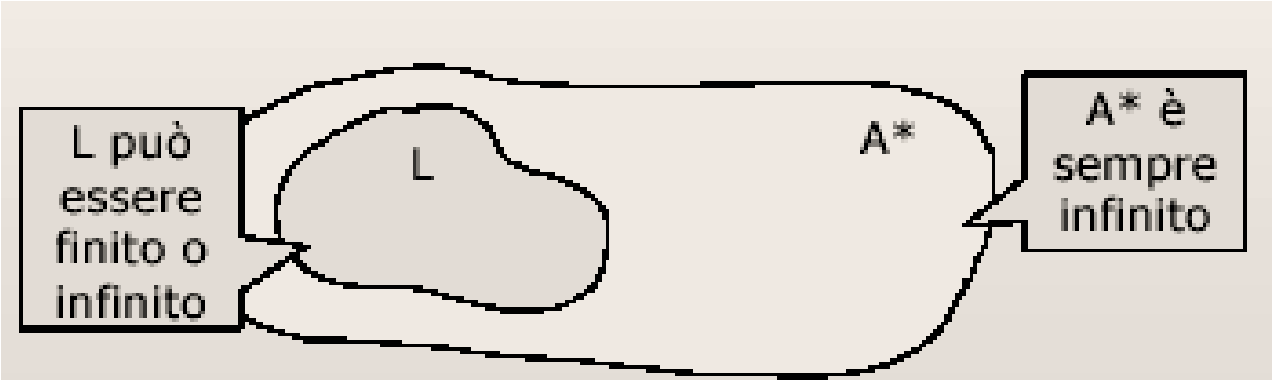
✓ **Cardinalità di un linguaggio**

il numero di frasi di quel linguaggio

- linguaggio finito: cardinalità finita
- linguaggio infinito: cardinalità infinita

✓ **Chiusura dell'alfabeto A**

linguaggio più grande dell'alfabeto A, quello che contiene tutte le stringhe dell'alfabeto



A* è l'insieme dei linguaggi su A

Specifiche del linguaggio dato l'alfabeto

- Se il linguaggio è ***finito***
  - basta elencare le sue frasi
- Se il linguaggio è ***infinito***
  - non possiamo elencare le frasi
  - bisogna trovare una notazione per descriverne tutte e sole le frasi del linguaggio
  - e ovviamente, tale notazione deve essere finita

Problema: Elencare tutte le frasi valide che sono in numero infinito

⇨ Rappresentare un numero infinito di casi mediante una descrizione finita

- *Soluzione:* ***algoritmo di enumerazione***

Oss.: eseguendo l'algoritmo (che è un iniseme finito di regole di calcolo) si produce l'enumerazione delle frasi del linguaggio: senza fine se il linguaggio ne contiene un numero illimitato.

Operazioni sui linguaggi

Siano L1 ed L2 linguaggi su un alfabeto Σ^*

- Unione $L1 \cup L2 = \{x \in \Sigma^* | x \in L1 \vee x \in L2\}$
- Intersezione $L1 \wedge L2 = \{x \in \Sigma^* | x \in L1 \wedge x \in L2\}$
- Complementazione $\overline{L1} = \{x \in \Sigma^* | x \notin L1\}$
- Concatenazione (prodotto) di linguaggi

$L1 \circ L2 = \{x \in \Sigma^* |$ esistono x1 ed x2 tali che $x1 \in L1 \wedge x2 \in L2$ e $x = x1 \circ x2\}$

Esempio L1= {Buona }, L2= {serata, notte},L3= {Luca, Paolo}

L1 ° L2 ° L3 = {Buona serata Luca, Buona notte Luca, Buona serata Paolo, Buona Notte Paolo}

Operazioni sui linguaggi

- potenza di un linguaggio

Dato un linguaggio L

$L^h = L \circ L^{h-1}$, $h \geq 1$ $L^0 = \{\varepsilon\}$ per convenzione.

Quindi: L1 = L, L2 = L°L ecc.

Esempio Sia $L1 = \{a^n b^n \mid n \geq 1\}$ $L2 = L \circ L = \{a^n b^n a^m b^m \mid n,m \geq 1\}$

$aaabbbaabb \in L2$, $aaabbbaaabbb \in L2$, $aaabbaabb \notin L2$

Operazioni sui linguaggi

- Iterazione di un linguaggio

Dato un linguaggio L

$$L^* = \bigcup_{h=0}^{\infty} L^h$$

Con $\varepsilon \in L^*$ per definizione

- Se si vuole indicare il linguaggio L* escludendo la stringa vuota si usa il simbolo L+

Esempi

L={ab, aab}, L^*={e, ab, aab, abab, abaab, aabab, aabaab, ecc.}

L={a, b}, L^* contiene tutte le stringhe definite sui due simboli a e b, inclusa la stringa vuota

L={aa, bb}, L^* contiene tutte le stringhe costituite da un numero pari di a e un numero pari di b, inclusa la stringa vuota

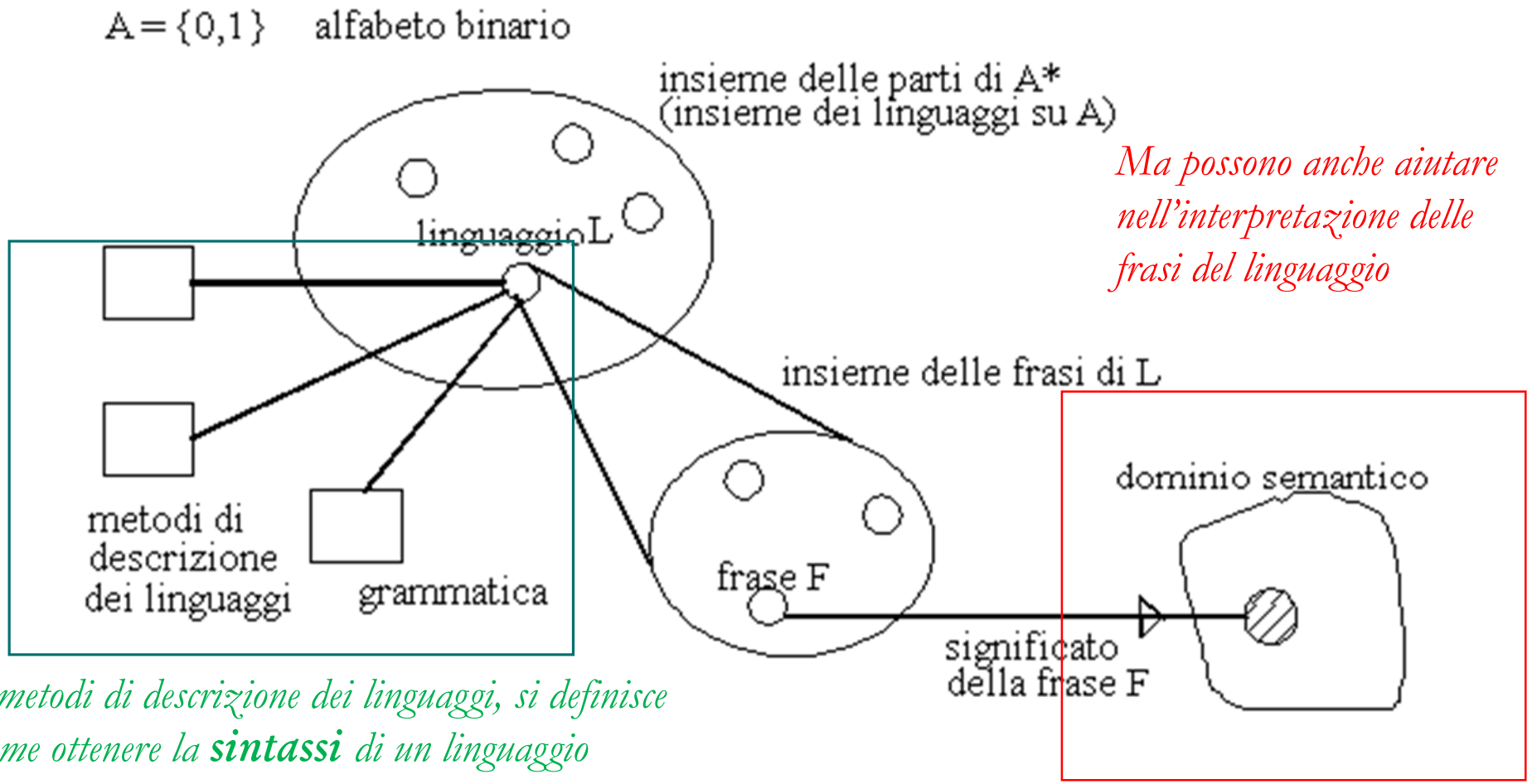
Definizione di un linguaggio

Vi sono diversi approcci:

- **Algebrico** il linguaggio è costruito a partire da linguaggi più elementari utilizzando operazioni su linguaggi
- **Generativo** si definisce una grammatica composta dalle regole strutturali che devono essere soddisfatte dalle stringhe del linguaggio
- **Riconoscitivo** si realizza una 'macchina' (un algoritmo di riconoscimento) che ricevendo una stringa in input dice se essa appartiene o no al linguaggio.

Oss.: Non tutti i linguaggi possono essere riconosciuti mediante programmi di calcolatore (o, equivalentemente, mediante algoritmi) e non tutti possono essere definiti mediante grammatiche.

Sintassi e Semantica di un linguaggio



Ma possono anche aiutare nell'interpretazione delle frasi del linguaggio

I metodi di descrizione dei linguaggi, si definisce come ottenere la ***sintassi*** *di un linguaggio*

http://www.dsi.unive.it/~salibra/Downloads/Linguaggi__descrizione_e_ricon/IMG00001.gif

Linguaggi formali

- **Espressioni regolari**
  - Approccio algebrico

Espressioni regolari

- *Quali linguaggi possono essere rappresentati mediante espressioni regolari?*

- I linguaggi rappresentabili con espressioni regolari sono una sottoclasse piccola (ma utile) di tutti i possibili linguaggi: ***la classe dei linguaggi regolari*** (linguaggi di tipo 3 nella classifica di Chomsky)

Oss.: *Per rappresentare espressioni aritmetiche o linguaggi di programmazione sono necessari linguaggi di tipo 2, più potenti: l'utilizzo dei linguaggi regolari permette di rappresentare, per esempio, un'algebra di Boole, ma non sono in grado di fare tutti i tipi di conteggi e per questo sono anche chiamati «non-counting»*

Linguaggi regolari *Osservazione*

Si possono descrivere con tutti i tre approcci descritti precedentemente:

- **Algebrico *Espressioni Regolari (RE)***
  - descrivono la struttura delle stringhe del linguaggio
- **Generativo *Grammatiche regolari (RG)***
  - definiscono un modello generativo delle stringhe
- **Riconoscitivo *Automi a Stati Finiti (FSA)***
  - Un automa definisce un riconoscitore per stabilire se una stringa appartiene ad un dato linguaggio regolare
- La scelta del formalismo dipende dallo scopo dell'applicazione

Espressioni regolari *nei linguaggi formali*

- Le espressioni regolari sono quindi un metodo per rappresentare linguaggi.
- Ad ogni espressione regolare ‘e’ corrisponde il linguaggio L(e) che essa rappresenta.
- La rappresentazione è basata sulle operazioni che ci permettono di definire L(e) a partire da linguaggi elementari.
- In una espressione regolare usiamo il simbolo **+** per indicare l’unione di linguaggi, il simbolo · (o semplicemente nessun simbolo) per indicare la concatenazione e il simbolo ***** per indicare l’iterazione.

Espressioni regolari *nei linguaggi formali*

- *Esempio*: si vuole rappresentare il linguaggio costituito da tutte le stringhe che

1. cominciano con *a*,
2. proseguono con un numero arbitrario (anche nullo) di *b*
3. e terminano con la stringa *bab* o con la stringa *aba*

- possiamo scrivere

ab(bab+aba)*

Espressioni regolari *nei linguaggi formali*

Dato un alfabeto Σ, si dice espressione regolare una stringa r sull'alfabeto

$(\Sigma \cup \{+,*,(,),.,\emptyset\})$

tale che:

1. $r = \emptyset$ *oppure*
2. $r \in \Sigma$ *oppure*
3. r = (s+t) *oppure* r = (s.t) *oppure* r = s*, dove s e t sono essere stesse espressioni regolari

- Il significato di essa (cioè il linguaggio L(e) da essa rappresentato) è ***definito induttivamente*** come segue:

espressione	linguaggio
Ø	insieme vuoto
a	{a} (per ogni simbolo a ∈ Σ)

(s+t), (s.t), s* rispettivamente L(s) ∪ L(t), L(s) ° L(t), L(s)*

Espressioni regolari *nei linguaggi formali*

Esempio:

Data l'espressione r = (((a.a) + b) . c*) abbiamo

L(r) = L(((a.a) + b) . c*) = L ((a.a) + b) ° L(c*)

in cui

L ((a.a) + b) = L (a.a) ∪ L(b) e L(c*) = (L(c))*.

Inoltre

L (a.a) = L (a) ° L (a)

Quindi

L(r) = ((L (a) ° L (a)) ∪ L(b)) ° (L(c))*

dove L(a) = {a}, L(b) = {b}, L(c) = {c} sono i linguaggi elementari costituiti da una sola stringa di un solo carattere.

Ossia L(r) = {aa, b, aac, bc, aacc, bcc, ...} cioè il linguaggio costituito da tutte le stringhe che iniziano con aa o b e proseguono con un numero arbitrario (anche nullo) di c.

Espressioni regolari *nei linguaggi formali*

Esempi

Scrivere l'espressione regolare sull'alfabeto {0,1}

1. che denota il linguaggio formato da tutte le stringhe che contengono "101" come sottostringa
    1. (0 + 1)* .101. (0 + 1)*
2. Tutte le sequenze alternate (cioè non contengono 00 o 11) di 0 e 1 che iniziano e finiscono per 1 o che iniziano e finiscono per 0
    1. (1.0)*1+(0.1)*0
3. Tutte le sequenze con un numero dispari di 1
    1. (1+101*0)*
    2. oppure 1*(01*01)*

Espressioni regolari: *Storia*

- Termine coniato dal matematico S. Kleene (‘50)
- Anni ‘60 entrarono nel mondo UNIX (editor QED di Thompson che metteva a disposizione, dalla sua interfaccia a riga di comando, il comando global regular expression print (grep))
- Negli anni ’80 furono implementate nel linguaggio PERL
- ***Sono un formidabile metodo formale di descrivere i pattern da trovare***
  - utilizzate principalmente per la ricerca e la sostituzione di porzioni del testo.

***Oss.**: Sono uno strumento importante nell’informatica teorica dove , per esempio, sono utilizzate per rappresentare tutti i possibili cammini su un grafo.*

Espressioni regolari: *Pattern Matching*

- Una ***espressione regolare*** è una sequenza di simboli che identifica un insieme di stringhe tramite l'utilizzo di «metacaratteri»
- Si può quindi ricercare in un testo il pattern rappresentato da un espressione regolare dove:
  - ogni carattere corrisponde a se stesso (quindi ritrova sé stesso) a meno che non si tratti di un metacarattere;
  - i metacaratteri sono quindi quei caratteri che cambiano il comportamento della corrispondenza di pattern:
    - ^ $ ( ) \ | @ [{ ? . + *

***Esempio.**: la stringa «pippo» trova tutte le occorrenze di pippo*
la stringa «.atto» trova ogni stringa di cinque caratteri come gatto, matto o patto

Espressioni regolari: *Pattern Matching*

Regole del gioco …

- Generalmente, le corrispondenze di pattern iniziano a sinistra della stringa obiettivo e procedono verso destra;
- Le corrispondenze di pattern restituiscono vero (in qualsiasi contesto) solo se l'intero pattern è presente nella stringa obiettivo;
- E' trovata per prima, la prima corrispondenza possibile (quella più a sinistra) nella stringa obiettivo.
  - Le espressioni regolari non lasciano indietro una buona corrispondenza per cercarne un'altra;
- Si prende la prima corrispondenza più grande possibile.
  - L'espressione regolare potrebbe trovare subito una corrispondenza e cercare di estenderla il più possibile.
  - Le espressioni regolari cercano di estendere quanto più possibile la corrispondenza.

Espressioni regolari: *Pattern Matching*

Sintassi per il Pattern Matching con espressioni regolari su base UNIX

- **[...]** per includere uno qualsiasi dei caratteri definiti in parentesi
  - E' possibile specificare singoli caratteri o intervalli di caratteri adiacenti
    - (es. A-Z per indicare tutte le lettere alfabetiche maiuscole)
    - es. [a-zABC] riconosce una qualsiasi lettera alfabetica minuscola oppure A, B, o C
- **[^...]** per escludere uno qualsiasi dei caratteri in parentesi
  - es. [^0-9] riconosce qualsiasi carattere non numerico
- **** -escape- per segnalare sequenze speciali o considerare caratteri speciali come caratteri normali
  - es. \? Cerca il ?

Espressioni regolari: *Pattern Matching*

Sintassi …su base UNIX

- simboli speciali per identificare un carattere ...
  - es. \d = numerico, ossia [0-9]
  - es. \D = non numerico, ossia [^0-9]
- | - or logico - per esprimere un'alternativa tra due espressioni
  - sintassi: regexp1 | regexp2
  - es. "A|B" riconosce sia il carattere A sia il carattere B
- . indica un carattere qualsiasi
  - es. "A.B" riconosce la stringa AoB e quella AuB, ma anche AOB
- ^ corrisponde all'inizio della stringa
  - sintassi: ^regexp
  - es. " ^parma" trova «Parma, domenica …» ma non «A parma , domenica…»B
- $ corrisponde alla fine della stringa

Espressioni regolari: *Pattern Matching*

Sintassi „„su base UNIX

- Usa le **(...)** per raggruppare espressioni e creare clausole complesse
  - es. ga(zz|tt)a trova sia gazza che gatta
- Si può specificare la numerosità dei composti con le graffe es. " es. "\d{3,5}" riconosce numeri composti da almeno tre cifre ed al massimo da cinque
- ***** indica zero o più occorrenze di un'espressione
  - es(ab)* riconosce sequenze di «ab» di qualsiasi lunghezza, come «ab» o «abab» ma anche la stringa vuota
- **+** indica una o più occorrenze di un'espressione
  - es(ab)+riconosce sequenze di «ab» di qualsiasi lunghezza, come «ab» o «abab»
- **?** indica zero o al più una occorrenza di un'espressione
  - es(ab)? riconosce sequenze di «ab» ma non «abab»

Espressioni regolari: *Pattern Matching*

Esempi

Validazione dei dati di un form

```
function verifica(d) {
  var expr=/^\d{2}-\d{2}-\d{4}$/
  if (expr.test(d))
    window.alert(d+": formato corretto")
  else
    window.alert(d+": formato errato")
}
```

```
<form>
  data (gg-mm-aaaa) <input type="text" name="data">
  <input type="button" value="verifica"
    onclick="verifica(data.value)">
</form>
```

checkdata.html

Espressioni regolari: *Pattern Matching*

Esempi

- Validare un nome a dominio

^[a-z0-9\-\.]+\.(it|com|org|net|eu|mobi)$

- Verificare che un file abbia una data estensione

^.+\.zip$

- Validazione di un email

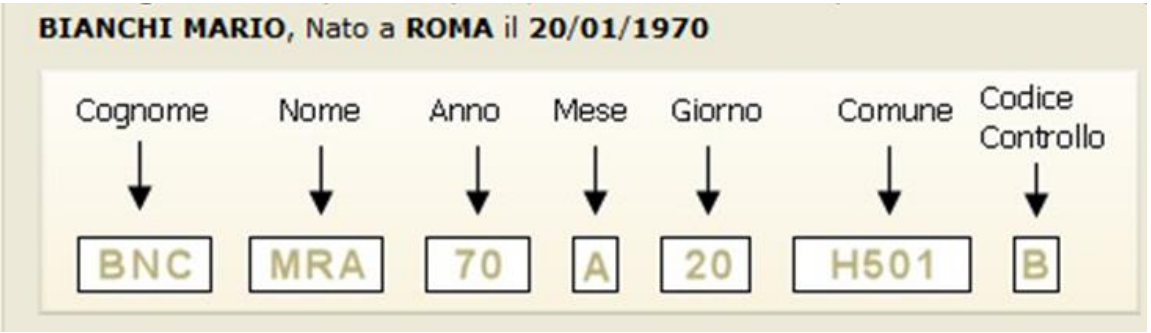
^[a-z0-9_]+@[a-z0-9\-]+\.[a-z0-9\-\.]+$]

Espressioni regolari: *Pattern Matching*

Esempi

- Validare un codice fiscale

^[a-z]{6}[0-9]{2}[a-z][0-9]{2}[a-z][0-9]{3}[a-z]$



Il Codice Fiscale e' costituito da **16 caratteri** alfanumerici, indicativi dei dati anagrafici della persona fisica, composti nel seguente modo:

3 caratteri alfabetici per il cognome;
3 caratteri alfabetici per il nome;
2 caratteri numerici per l'anno di nascita;
1 carattere alfabetico per il mese di nascita;
2 caratteri numerici per il giorno di nascita ed il sesso;
4 caratteri associati al Comune oppure allo Stato estero di nascita.
1 carattere alfabetico usato come carattere di controllo

Espressioni regolari: *AWK*

- E' un linguaggio di programmazione interpretato ***orientato alla manipolazione di dati di tipo testuale***, sia in forma di *file* che di flusso di dati provenienti dallo *standard input*
  - https://it.wikipedia.org/wiki/Awk
- Implementazione GNU, detta ***gawk***
- Rappresenta un filtro generico per files di testo basato su pattern-matching e programmabile con un linguaggio molto simile al C ma di utilizzo molto più semplificato.
- Elabora una riga alla volta dei files di input, eseguendo azioni diverse a seconda che la riga stessa soddisfi certe condizioni o contenga certi patterns.

  pattern { azione }

  pattern { azione }

  ...

Espressioni regolari: *AWK*

- Un programma AWK è costituito da una serie di coppie pattern-azione
- Un pattern può essere costituito da:
  - ***/espressione regolare/***
    - il pattern risulta vero per tutte le righe che soddisfano l'espressione regolare in questione
  - ***espressione***
    - una qualsiasi espressione C risulta vera se il suo risultato finale è diverso da zero e dalla stringa vuota.
  - ***pattern vuoto***
    - un pattern vuoto, cioè mancante, è soddisfatto da qualsiasi riga di input e pertanto la corrispondente azione viene sempre eseguita, a meno che una delle azioni precedenti decida di saltare al record successivo.

Espressioni regolari: *AWK*

ESEMPIO

- Regola che dice di stampare la riga di input quando questa contiene la stringa 'MARIO'. Tutto ciò che non contiene MARIO viene letto ma non stampato in output

  - cat elenco.txt | awk '/MARIO/ { print }'

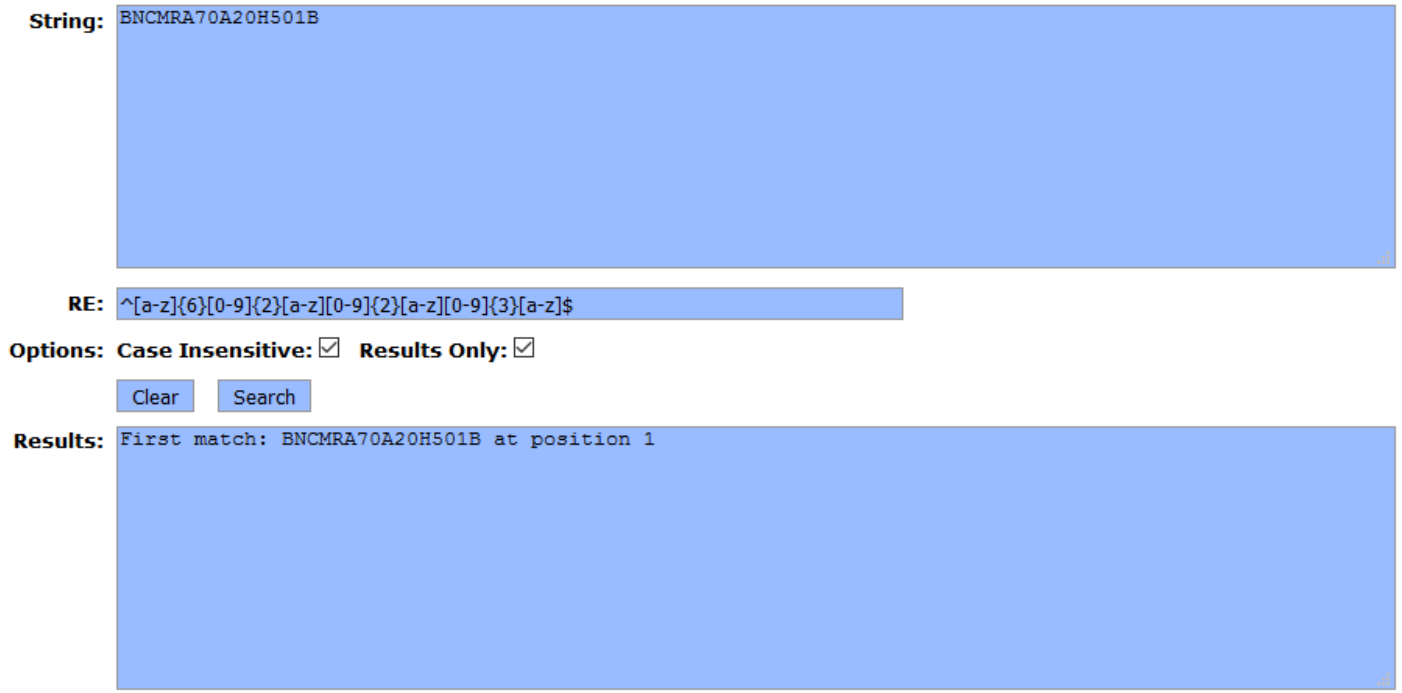
  oppure semplicemente

  - cat elenco.txt | awk '/MARIO/‘

Calcolo espressioni regolari online

regex

- http://www.zytrax.com/tech/web/regex.htm
- il Test (in javascript)
- http://www.zytrax.com/tech/web/regex.htm#experiment



Calcolo espressioni regolari online

- Altri:
- https://regex101.com/

- In questo sono salvate le «Top Regular Expressions»
- *https://www.regexpal.com/*

- Questo sistema funziona come un generatore di espressioni regolari. Invece di provare a creare l'espressione regolare, si inizia con la stringa che si desidera cercare
- https://txt2re.com/

- Un «Ruby regular expression editor» con rferimento ad un «Programming Ruby pragmatice user guide»
- http://rubular.com/