Corso di Laurea Magistrale in Ingegneria Informatica
A.A. 2011-2012

Linguaggi Formali e Compilatori

Linguaggi, grammatiche, espressioni regolari

Giacomo PISCITELLI

Modello formale di un linguaggio

Un **linguaggio formale** può essere definito in una grande varietà di modi:

- l'**insieme delle stringhe derivate** nell'ambito di una grammatica formale (cfr. gerarchia di Chomsky);
- l'**insieme delle stringhe fornite da un'espressione regolare** (un caso particolare ed importante del punto precedente);
- l'**insieme delle stringhe accettate da qualche automa**, come una macchina di Turing o un automa a stati finiti;
- le **domande a risposta affermativa**, nell'ambito di un insieme di domande la cui risposta può essere solo SI o NO.

In generale diremo che un **modello formale** che può riconoscere e generare tutte e sole le stringhe di un linguaggio formale agisce come una definizione di tale linguaggio.

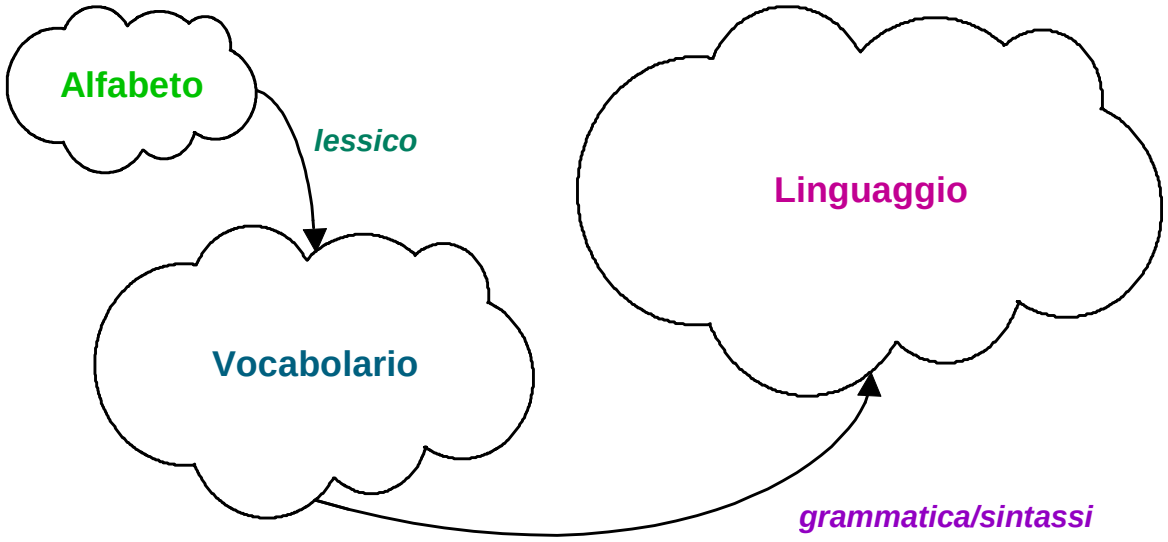
Secondo i due principali approcci alla definizione dei linguaggi formali, un modello si può concretizzare in una **grammatica formale** (*approccio generativo*), o in un **automa** (*approccio riconoscitivo*).

Grammatiche generative

Il termine **grammatica generativa** (*generative grammar*) si riferisce ad un approccio tratto dalla teoria della dimostrazione per lo studio della sintassi, parzialmente ispirato dalla teoria della grammatica formale (*formal grammar*) e inaugurato da Noam Chomsky.

Una *grammatica generativa è un insieme di regole che "specificano" o "generano" in modo ricorsivo* (cioè per mezzo di un "sistema di riscrittura") le espressioni ben formate (*well-formed expressions*) di un linguaggio.

Questa definizione include un gran numero di diversi approcci alla grammatica. Nel caso dei linguaggi automatici (di programmazione):

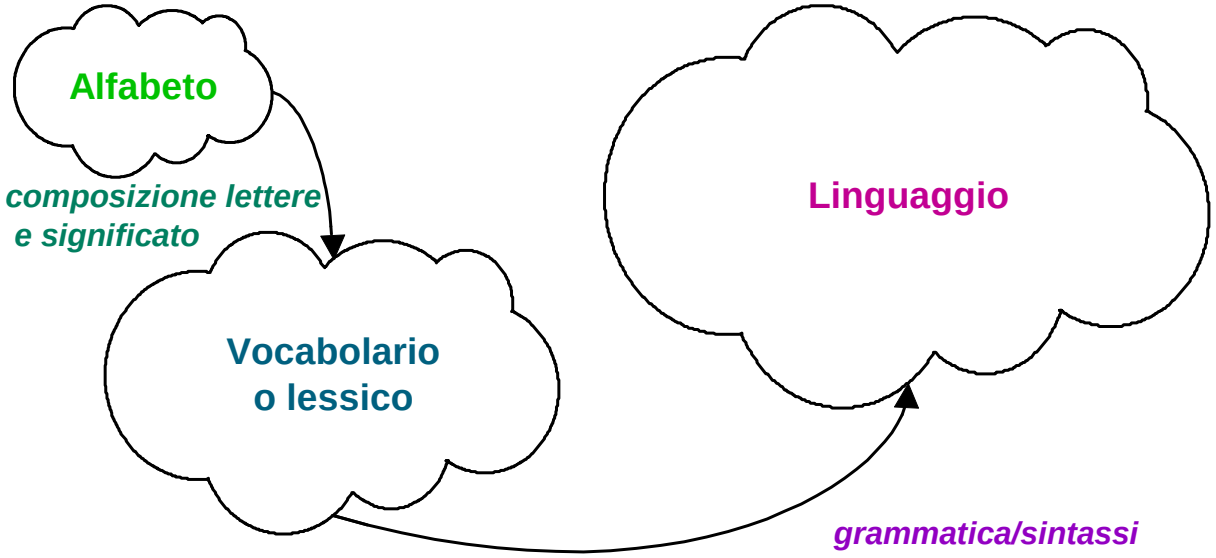


Grammatiche generative nella linguistica

Il termine grammatica generativa è anche largamente usato nella linguistica, all'interno della quale questo tipo di grammatica formale gioca un ruolo cruciale.

In realtà la grammatica generativa dovrebbe essere distinta dalla grammatica tradizionale, che solitamente:

- è fortemente prescrittiva, anziché puramente descrittiva;
- non è matematicamente esplicita;
- storicamente ha analizzato un insieme relativamente ristretto di fenomeni sintattici.



Grammatiche generative nella linguistica

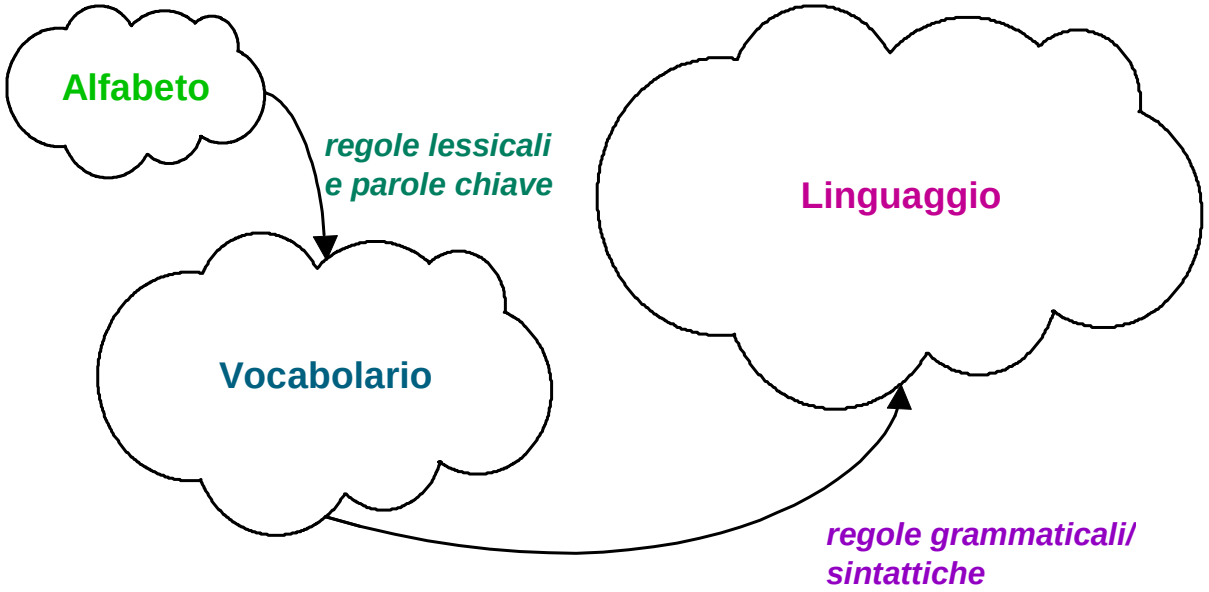
Il termine *grammatica generativa* può anche riferirsi ad un particolare insieme di regole formali per un particolare linguaggio naturale. Ad esempio si può parlare di una *grammatica generativa dell'italiano*. Essa, però, non rispetta rigorosamente i dispositivi formali di una grammatica generativa in senso stretto.

Infatti una grammatica generativa in senso stretto è un dispositivo formale che può enumerare ("generare") tutte e sole le frasi di un linguaggio.

In sostanza, una grammatica generativa è un algoritmo che può essere usato solo per decidere se una certa frase è o meno grammaticalmente ben formata.

Tale caratteristica di una grammatica non si attaglia completamente ai linguaggi naturali, mentre è sostanzialmente rispettata dai linguaggi automatici.

Grammatiche generative formali



Per altro verso, nella maggior parte dei casi, una grammatica generativa è in grado di generare un numero infinito di "stringhe" a partire da un insieme finito di regole.

Queste proprietà sono desiderabili per un modello del linguaggio naturale. Infatti gli esseri umani, pur possedendo un cervello di capacità finita, possono generare e comprendere una enorme quantità di frasi distinte.

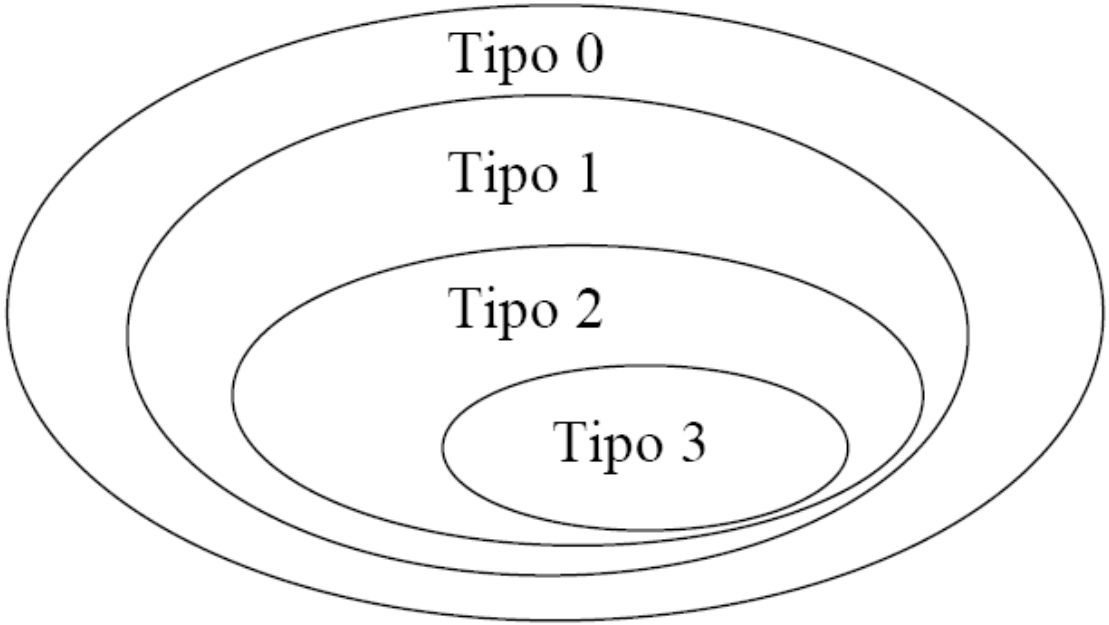
È per questo che alcuni linguisti sostengono che l'insieme delle frasi grammaticali di un linguaggio naturale è veramente infinito e sono propensi ad assumere la completa applicabilità delle grammatiche generative anche ai linguaggi naturali.

Gerarchia di Chomsky

Le grammatiche generative possono essere descritte e confrontate fra loro con l'aiuto della **Gerarchia di Chomsky**, proposta da lui negli anni '50. Questa definisce una serie di tipi di grammatiche formali aventi potere espressivo crescente.

Teoria degli automi: linguaggi formali e grammatiche formali			
gerarchia di Chomsky	**Grammatiche**	**Linguaggi**	**automa minimo**
Tipo-0	Generali	Ricorsivamente enumerabili	Macchina di Turing
Tipo-1	Dipendenti dal contesto	Sensibili al contesto	Lineare-limitato
Tipo-2	**Libere dal contesto**	Liberi dal contesto	**Automa a pila**
Tipo-3	**Lineari (o Regolari)**	Lineari (o Regolari)	**Automa stati finiti**

Chomsky Hierarchy



Ciascuna categoria di linguaggio o grammatica è un sottoinsieme del proprio sovrainsieme di categoria

Chomsky Hierarchy

Quando la grammatica generativa fu proposta per la prima volta, fu unanimemente salutata come un modo di formalizzare l'insieme di regole implicite che una persona "conosce" quando conosce la propria lingua madre ed è in grado di produrre delle espressioni in essa.

Tuttavia Chomsky ha ripetutamente respinto quella interpretazione. Secondo lui, la grammatica di una lingua è l'esposizione di ciò che una persona deve conoscere al fine di riconoscere una espressione come grammaticale, ma non una ipotesi sul processo coinvolto nella comprensione o nella produzione del linguaggio.

In ogni caso la realtà è che la maggior parte dei madrelingua respingerebbero molte frasi prodotte anche da una grammatica a struttura sintagmatica, che preveda, cioè, unità sintattiche autonome. Ad esempio, sebbene la grammatica consenta annidamenti molto profondi, le frasi con annidamenti profondi non sono accettate da chi le sente pronunciare, e il limite di accettabilità è una faccenda empirica che varia da individuo ad individuo, non qualcosa che può essere facilmente catturato in una grammatica formale.

Grammatiche generative

Formalmente una **grammatica generativa** G è una quadrupla (T, N, S, P) composta da:

- T: alfabeto dei simboli terminali, cioè l'insieme finito dei simboli terminali
- N: alfabeto dei simboli non terminali
- S: *simbolo distintivo* o *scopo* o ancora *assioma* della grammatica (appartiene a N)
- P: insieme di coppie (v, w) di stringhe dette produzioni costruite sull'unione dei due alfabeti, denotate anche con $v \rightarrow w$

La stringa v contiene almeno un simbolo non terminale.

Normalmente $T \cap N = \phi$.

Ogni simbolo della grammatica è un elemento di $T \cup N$.

Espressioni regolari

Una volta introdotta la struttura generale di una grammatica generativa, ricordiamo che l'interesse per tale specificazione nasce dalla necessità di individuare un insieme limitato di regole capaci di generare tutte le frasi (eventualmente infinite) di un linguaggio automatico e, in particolare, tutti i lessemi consentiti da un linguaggio.

Una importante notazione per indicare le regole capaci di specificare la struttura dei lessemi è costituita dalle espressioni regolari.

Pur non essendo le espressioni regolari in grado di esprimere tutte le possibili strutture dei lessemi di un linguaggio, esse sono molto efficaci per specificare i tipi di strutture di cui effettivamente si avvalgono i linguaggi di programmazione per costruire i tipici token che l'analizzatore lessicale è chiamato a riconoscere.

Dopo aver studiato la notazione formale delle espressioni regolari, passeremo a studiare i generatori di analizzatori lessicali (quali LEX) nei quali tali espressioni regolari vengono usate per generare automi in grado di riconoscere specifici token.

Dai caratteri alle espressioni regolari

Dato un alfabeto Σ –costituito, per esempio, dai caratteri {a,b,c,…,z }– possiamo immaginare che Σ sia un linguaggio le cui stringhe siano tutte di lunghezza 1. In tal caso è possibile costruire, a partire da L, altri linguaggi, facendo uso degli operatori unione $\Sigma \cup \Sigma$, concatenazione $\Sigma.\Sigma$, stella di Kleene Σ^*, croce Σ^+. Le stringhe di tali linguaggi potranno essere di lunghezza 2, 3, … .

Questo processo costruttivo delle stringhe è stato considerato così utile da suggerire l'introduzione di una notazione, detta espressione regolare, che descrive tutti i linguaggi che possono essere costruiti dagli operatori unione, concatenazione, stella di Kleene e croce applicati ai caratteri (o simboli) di un qualche alfabeto.

Definizione di espressione regolare

Una **espressione regolare** r – definita a partire da un alfabeto Σ e da un insieme di metasimboli {+,*,(,),.,Ø, ε}, supposti non appartenere a Σ – è una stringa r appartenente all'alfabeto (Σ∪{+,*,(,),.,Ø}), tale che valga una delle seguenti condizioni:

1. r = Ø
2. r = ε
3. r = x, dove x è un qualunque carattere ∈ Σ
4. r = (s+t) ove s e t sono espressioni regolari su Σ e + è l'operatore unione
5. r = s.t ove s e t sono espressioni regolari su Σ e + è l'operatore concatenazione
6. r = s^* ove s è un'espressione regolare su Σ e * è l'operatore chiusura di Kleene

Esempi di espressioni regolari

Spesso nelle espressioni regolari

st è sinonimo di s.t

s | t è sinonimo di s+t

$\varepsilon = \emptyset^*$ $r^h = r \ldots r$ (h volte)

e la precedenza fra gli operatori $*$, ., + prevede che  * > . > +

(a+b)* rappresenta il linguaggio $L=(\{a\}\cup\{b\})^*$

(a+b)*a rappresenta il linguaggio $L=\{x|\ x \in (\{a\}\cup\{b\})^*$ e "x termina con a"}

(a* + ((a.a) . b*)) = a* + aab*

Implicazione di espressioni regolari

L'espressione regolare r implica (o genera) l'espressione s (notazione $r \rightarrow s$) se:

- $r = x\alpha y$
- $s = x\beta y$
- le espressioni x, y, α sono sottoespressioni di r e vale una delle seguenti condizioni:
  1. $\alpha = e_1 \cup \ldots e_k \cup \ldots e_n$ e $\beta = e_k$
  2. $\alpha = e^*$ e $\beta = e \ldots e$ (h volte)

Proprietà dell'implicazione:

$r \rightarrow^n s$ (r implica s in n passi) se $r = e_0 \rightarrow e_1 \ldots e_{n-1} \rightarrow e_n = s$

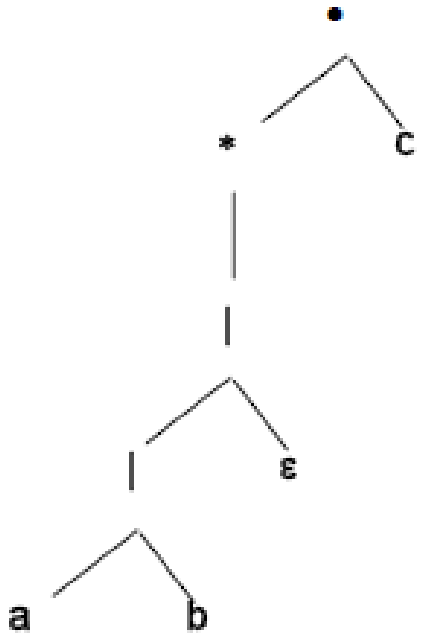
$r \rightarrow^* s$ (r implica riflessivamente e transitivamente s) se $r \rightarrow^n s$ per qualche $n \geq 0$

$r \rightarrow^+ s$ (r implica transitivamente s) se $r \rightarrow^n s$ per qualche $n \geq 1$

- ✓ $(ab)^* \cup bca \rightarrow (ab)^* \rightarrow ababab$
- ✓ $(ab)^* \cup bca \rightarrow^2 ababab$
- ✓ $ababab \rightarrow^0 ababab$

Espressioni regolari e alberi equivalenti

Data l'espressione regolare **r: ((a | b) | ε)* c**, si chiede di rappresentare l'albero dell'espressione r.



Grammatiche regolari o lineari (tipo 3)

Una sottoclasse utile di linguaggi è generata dalle **grammatiche lineari o regolari**.

Una grammatica regolare (lineare o di tipo 3) è denotata da G = (V, Σ, P,S), dove V è l'insieme dei non terminali, Σ è l'insieme dei terminali, S è l'assioma, P è un insieme finito di produzioni, con ogni produzione nella forma:

$$A \rightarrow \beta,$$

con:

$$A \in V \qquad \text{e} \qquad \beta \in (V \cup \Sigma)^*$$

Ogni produzione di una grammatica regolare può essere del tipo:

$A \rightarrow aB$, $B \rightarrow b$ (grammatica lineare destra)

oppure

$B \rightarrow Ab$, $A \rightarrow a$ (grammatica lineare sinistra)

ove

$$A, B \in V \qquad \text{e} \qquad a, b \in \Sigma$$

Cosa sono i linguaggi regolari?

Sono tutti e soli i linguaggi che si possono generare con grammatiche generative di tipo 3.

Un tale linguaggio può essere espresso mediante le operazioni di concatenamento, unione e stella di Kleene applicate un numero finito di volte a linguaggi unitari e al linguaggio vuoto.

Più rigorosamente, un **linguaggio regolare** è un linguaggio le cui stringhe sono implicate da un'**espressione regolare**.

Il linguaggio regolare $L(r)$ generato dall'espressione r è l'insieme delle stringhe implicate da r

$$L(r) \equiv \{x \mid r \rightarrow^* x\}$$

$L((ab)^* \cup bca) \equiv \{\varepsilon, ab, bca, abab, ababab, \ldots\}$

In definitiva un linguaggio è detto regolare se è prodotto da un'**espressione regolare**, cioè dall'uso degli operatori linguistici di unione, concatenazione e chiusura applicati ai simboli dell'alfabeto alla base di un linguaggio.

La famiglia dei linguaggi regolari è la più piccola famiglia di linguaggi che

- ✓ contiene tutti i linguaggi finiti
- ✓ è chiusa rispetto a concatenamento, unione e stella

Grammatiche regolari o lineari (tipo 3)

Esempio di grammatica regolare
V = {S}
Σ = {a, b}
P = {S → ε (stringa vuota)
S → abS}
S = {S}

Stringhe appartenenti al linguaggio:
S → abS → ab

S → abS → ababS → abab

Si noti che in ogni parte destra delle produzioni:

- è contenuto al più un non terminale;
- il solo non terminale sempre si trova alla fine della regola (o alternativamente all'inizio).

Le grammatiche di tipo 3 sono molto restrittive (non si possono utilizzare per la sintassi dei linguaggi di programmazione) **ma i loro riconoscitori sono molto efficienti**.

L'importanza delle espressioni regolari

Sebbene fossero state formalizzate già fin dagli anni '40, le espressioni regolari entrarono nel mondo informatico alla fine degli anni '60, in ambiente Unix: il primo editor di testo che implementava delle funzioni che ne permettessero l'uso fu una versione di QED scritta da Ken Thompson, uno degli autori di Unix.

L'editor metteva a disposizione un comando chiamato *global regular expression print*, che successivamente fu reso un applicativo indipendente, **`grep`**.

Le espressioni regolari non ebbero grande diffusione ed utilizzo fino agli anni '80, quando fu inventato il linguaggio di programmazione **Perl** che permetteva nativamente l'uso di espressioni regolari.

La versatilità del linguaggio, dovuta anche al fatto d'essere un linguaggio interpretato, ne permise l'utilizzo in svariate situazioni e favorì lo sviluppo del formalismo di Perl per le espressioni regolari, che diventò *de facto* uno standard.

La grandissima diffusione di questi strumenti spinse alcuni sviluppatori a implementare le espressioni regolari anche in altri linguaggi, a mezzo di librerie, o persino a svilupparne implementazioni native per alcuni linguaggi, come Java.

Attualmente le espressioni regolari sono disponibili nella maggioranza degli editor di testo per GNU/Linux e altri sistemi Unix-like.

Impiego delle espressioni regolari

Come vedremo, le espressioni regolari sono principalmente utilizzate, nell'analizzatore lessicale, per definire come riconoscere, nell'ambito di un'istruzione di un programma, un lessema e, quindi, un token della frase.

Le espressioni regolari sono inoltre utilizzate dagli editor di testo per la ricerca e la sostituzione di porzioni del testo.

Grande importanza rivestono anche nell'informatica teorica, nella quale, ad esempio, sono utilizzate per rappresentare tutti i possibili cammini su un grafo.

Tuttavia i linguaggi di tipo 3 e, quindi, le espressioni regolari, sono adatte a rappresentare un ristrettissimo insieme di linguaggi formali (se volessimo rappresentare espressioni aritmetiche o linguaggi di programmazione, avremmo già bisogno di utilizzare linguaggi di tipo 2).

Espressioni regolari tradizionali di UNIX

La sintassi di base delle espressioni regolari in UNIX è stata ora definita obsoleta dal **POSIX**, ma è comunque molto usata a causa della sua diffusione. La maggior parte dei programmi che utilizzano le *regular expressions*, come `grep` e `sed`, usano il vecchio sistema.

In questa sintassi, la maggior parte dei caratteri sono visti come letterali, e trovano solo se stessi (**"a"** trova **"a"**, **"bc)"** trova **"bc)"** ecc.). Le eccezioni a questa regola sono i **metacaratteri**.

Nel seguito si riportano i più comuni metacaratteri, con l'avvertenza che quelli adottati nei sistemi operativi UNIX-like e quelli che saranno usati in seguito nell'analizzatore lessicale LEX presentano alcune lievi differenze.

I metacaratteri (1/2)

`.` Trova ogni singolo carattere

`[` `]` Trova un singolo carattere contenuto nelle parentesi. Ad esempio, **[abc]** trova o **una "a"**, **"b"**, o **"c"**. **[a-z]** è un intervallo e trova ogni lettere minuscola dell'alfabeto. Possono esserci casi misti: **[abcq-z]** trova **a**, **b**, **c**, **q**, **r**, **s**, **t**, **u**, **v**, **w**, **x**, **y**, **z**, esattamente come **[a-cq-z]**.

Il carattere '-' è letterale solo se è primo o ultimo carattere nelle parentesi: [abc-] o [-abc]. Per trovare un carattere '[' o ']', il modo più semplice è metterli primi all'interno delle parentesi: [][ab] trova ']', '[', 'a' o 'b'.

`[^` `]` Trova ogni singolo carattere non incluso nelle parentesi. Ad esempio, [^abc] trova ogni carattere diverso da "a", "b", o "c". [^a-z] trova ogni singolo carattere che non sia una lettera minuscola. Come sopra, questi due metodi possono essere usati insieme.

`^` Corrisponde all'inizio d'una riga (o ad una riga, quando usato in modalità multilinea)

`$` Corrisponde alla fine d'una riga (o ad una riga, quando usato in modalità multilinea)

I metacaratteri (2/2)

( ) Definisce una "sottoespressione marcata". Ciò che è incluso nell'espressione, può essere richiamato in seguito. Vedi **\n**.

\n Dove ***n*** è una cifra da 1 a 9; trova ciò che la *n*esima sottoespressione ha trovato. Tale costrutto è teoricamente **irregolare**.

*****
- Un'espressione costituita da un singolo carattere seguito da "*****", trova zero o più copie di tale espressione. Ad esempio, "**[xyz]***" trova "", "**x**", "**y**", "**zx**", "**zyx**", e così via.
- **\n***, dove ***n*** è una cifra da 1 a 9, trova zero o più iterazioni di ciò che la *n*esima sottoespressione ha trovato. Ad esempio, "**(a.)c\1***" trova "**abcab**" e "**abcaba**" ma non "**abcac**".
- Un'espressione racchiusa tra "**\(**" e "**\)**" seguita da "*****" non è valida. In alcuni casi trova zero o più ripetizioni della stringa che l'espressione racchiusa ha trovato. In altri casi trova ciò che l'espressione racchiusa ha trovato, seguita da un letterale "*****".

{*x*,*y*} Trova l'ultimo "blocco" almeno ***x*** volte e non più di ***y*** volte. Ad esempio, "**a{3,5}**" trova "**aaa**", "**aaaa**" o "**aaaaa**".

Esempi:

"`.atto`" trova ogni stringa di cinque caratteri come *gatto*, *matto* o *patto*

"`[gm]atto`" trova *gatto* e *matto*

"`[^p]atto`" trova tutte le combinazioni dell'espressione "`.atto` " tranne *patto*

"`^[gm]atto`" trova *gatto* e *matto* ma solo all'inizio di una riga

"`[gm]atto$`" trova *gatto* e *matto* ma solo alla fine di una riga