


# Appunti di Linguaggi Formali e Compilatori

Matteo Gianello

12 gennaio 2015

Quest'opera è stata rilasciata con licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale. Per leggere una copia della licenza visita il sito web <http://creativecommons.org/licenses/by-nc-sa/4.0/>. .

# Indice

<b>1</b>	<b>Introduzione alla teoria dei linguaggi formali</b>	<b>3</b>
1.1	Operazioni sulle stringhe . . . . .	4
1.2	Operazioni sui linguaggi . . . . .	5
<b>2</b>	<b>Linguaggi ed espressioni regolari</b>	<b>8</b>
2.1	La famiglia dei linguaggi regolari . . . . .	8
2.2	Sottoespressione di un'espressione regolare . . . . .	9

# Introduzione

Prima di entrare nel dettaglio della nostra trattazione definiamo che cos'è un *Linguaggio Formale*. Innanzitutto dobbiamo distinguere tra un *linguaggio natura* ed uno *artificiale*, un linguaggio naturale è quello che utilizziamo tutti i giorni e che si basa sul significato delle parole e soprattutto non possiede una struttura formale. Un linguaggio artificiale, invece, è quello utilizzato per la comunicazione tra macchine, è di tipo non verbale ma soprattutto è formale.

Un linguaggio si definisce **formale** se la sua *sintassi* (ovvero la sua struttura) e la sua *semantica* (ovvero il suo significato) sono definiti da precisi algoritmi. Questo permette di definire delle *procedure* precise per determinare la correttezza di una frase che appartiene ad un linguaggio e il suo significato. In un contesto più ristretto un **linguaggio formale** è un *oggetto matematico* costruito su di un *alfabeto* tramite regole assiomatiche chiamate *grammatiche* o tramite strumenti matematici chiamati *automi*.

## 1 Introduzione alla teoria dei linguaggi formali

Vediamo ora alcune definizioni che ci saranno utili durante il corso:

**Alfabeto:** insieme *finito* di elementi

$$\Sigma = \{a_1, a_2, a_3, \dots, a_k\}$$

**Cardinalità di un alfabeto:** è il numero di elementi che lo compongono

$$|\Sigma| = k$$

**Stringa:** insieme ordinato di elementi dell'alfabeto che possono essere anche ripetuti.

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

$$\text{Stringe} : abc \vee aab \vee ac \vee bbb$$

**Linguaggio:** insieme finito o infinito di stringhe di un alfabeto

$$L_1 = \{ab, ac, abc\}$$

$$L_2 = \{bc, bbc, \dots\}$$

La struttura insiemistica del linguaggio formale ha due livelli di profondità:

- Insieme non ordinato di elementi non atomici (*stringhe*) che sono
- sequenze ordinate di elementi atomici (*elementi o terminali*).

**Cardinalità di un linguaggio:** può essere finita o infinita.

$$|L_1| = |\{ab, ac, abc\}| = 3$$

**Lunghezza di una stringa:** indica il numero dei suoi elementi

$$|abb| = 3 \quad |abbc| = 4$$

**Uguaglianza tra stringhe:** due stringhe si definiscono uguali se e solo se hanno la stessa lunghezza, ed i suoi elementi coincidono ordinatamente da sinistra a destra.

$$x = a_1 a_2 \dots a_h \quad y = b_1 b_2 \dots b_k$$

$$x = y \iff h = k \wedge a_i = b_i \quad \forall i = 1 \rightarrow h$$

$x = abccbc$ <i>prefissi</i> : $a, ab, abc, abcc, abccb, abccbc$ <i>suffissi</i> : $c, bc, cbc, ccbc, bccbc, abccbc$ <i>sottostringhe</i> : $....., bc, cc, cb, ...$
---

Figura 1: Esempio di sottostringhe con prefissi e suffissi

## 1.1 Operazioni sulle stringhe

La *concatenazione* è l'operazione fondamentale con le stringhe, si tratta di fare il prodotto tra esse. Questa operazione è un'operazione di base e non fa altro che disporre in fila le due stringhe.

$$x = a_1 a_2 \dots a_h \quad y = b_1 b_2 \dots b_k$$

$$x \cdot y = a_1 a_2 \dots a_h b_1 b_2 \dots b_k = xy$$

Questo tipo di operazione gode della proprietà associativa e va ad influire sulla lunghezza della stringa come vediamo dalla 1 e dalla 2.

$$(xy)z = x(yz) \tag{1}$$

$$|xyz| = |x| + |y| + |z| \tag{2}$$

La concatenazione ci permette di introdurre un altro concetto relativo alle stringhe, ovvero quello della *stringa vuota* o *nulla* indicata con  $\varepsilon$  questa stringa è l'elemento neutro rispetto all'operazione di concatenazione, infatti, concatenando  $\varepsilon$  a destra o a sinistra di una stringa la stringa non cambia.

$$\varepsilon x = x\varepsilon = x$$

Tuttavia bisogna prestare attenzione a non confondere la stringa nulla con l'insieme vuoto  $\emptyset$ . Una volta capita l'operazione di concatenazione si possono individuare in una stringa delle *sotto-stringhe*, ovvero stringhe più piccole concatenate tra loro per formare una stringa più grande. Ad esempio data una stringa  $x$  essa può essere vista come un insieme di sotto-stringhe concatenate

$$x = uvv$$

Dove:

- $y$  è detta sotto-stringa.
- $u$  è chiamato prefisso.
- $v$  è chiamato suffisso.

Un esempio di suffissi, prefissi e sotto-stringhe è mostrato in Figura 1.

Si definiscono *sotto-stringhe proprie* tutte quelle stringhe che hanno  $u \neq \varepsilon$  e  $v \neq \varepsilon$ .

La *riflessione* è l'operazione che inverte l'ordine dei caratteri che compongono una stringa. Da quanto si vede in Figura2 notiamo che la riflessione gode di alcune proprietà, la riflessione di una stringa riflessa ci riporta alla stringa originale, la riflessione di due stringhe concatenate è uguale alla riflessione delle singole stringhe e successivamente la loro concatenazione, infine la riflessione della stringa vuota è uguale ancora alla stringa vuota. La *ripetizione* o anche *iterazione* è

$$\begin{aligned}
x &= a_1 a_2 \dots a_h \\
x^R &= a_h a_{h-1} \dots a_2 a_1 \\
(x^R)^R &= x \\
(xy)^R &= y^R x^R \\
\varepsilon^R &= \varepsilon
\end{aligned}$$

Figura 2: Proprietà della riflessione

$$\begin{aligned}
x &= ab \quad x^0 = \varepsilon \quad x^1 = x = ab \quad x^2 = (ab)^2 = abab \\
y &= a^3 = aaa \quad y^3 = a^3 a^3 a^3 = a^9 \\
\varepsilon^0 &= \varepsilon \quad \varepsilon^2 = \varepsilon
\end{aligned}$$

Figura 3: Esempio di ripetizione

quell'operazione che permette di concatenare più volte una stringa con se stessa; dato un indice  $m$  maggiore di uno allora la ripetizione non fa altro che concatenare  $m$ -volte la stringa con se stessa come si vede dalla Figura3. Come nel caso dell'algebra classica anche qui abbiamo una precedenza tra le operazioni, in particolare la *ripetizione* ha precedenza sulla *concatenazione*, anche la *riflessione* ha precedenza sulla *concatenazione*.

## 1.2 Operazioni sui linguaggi

Prima di tutto dobbiamo definire che cosa vuol dire effettuare un'operazione su di un linguaggio. In realtà effettuare un'operazione su di un linguaggio significa effettuare tale operazione su tutte le stringhe appartenenti a quel linguaggio. Ad esempio effettuare la riflessione di un linguaggio significa:

$$L^R = \{x \mid x = y^R \wedge y \in L\}$$

Ovvero si viene a creare un nuovo linguaggio di stringhe  $x$  che non sono altro che la riflessione di tutte le stringhe  $y$  del linguaggio di partenza.

Un altro esempio è il linguaggio definito come *prefisso*( $L$ ) ovvero:

$$\text{prefisso}(L) = \{y \mid x = yz \wedge x \in L \wedge y, z \neq \varepsilon\}$$

Da questa definizione possiamo definire un linguaggio *prefix-free* nel quale non esiste una stringa appartenente a questo linguaggio che sia prefisso di un'altra stringa dello stesso linguaggio.

$$L_1 = \{x \mid x = a^n b^n \wedge n \geq 1\}$$

Ad esempio:

$$a^2 b^2 \in L_1 \quad a^2 b \notin L_1$$

Passiamo ora a definire le *operazioni binarie* ovvero quelle composte da due operandi come la *concatenazione* tra due linguaggi.

$$L' \cdot L'' = \{xy | x \in L' \wedge y \in L''\}$$

Da questa definizione possiamo anche dimostrare la ripetizione di un linguaggio tramite la 3

$$\begin{cases} L^m = L^{m-1} \cdot L & \text{se } m > 0 \\ L^m = \{\varepsilon\} & \text{se } m = 0 \end{cases} \quad (3)$$

Una cosa molto importante da ricordare è la differenza tra la stringa nulla  $\varepsilon$  e il linguaggio vuoto  $\emptyset$ , infatti abbiamo che:

$$L \cdot \emptyset = \emptyset \cdot L = \emptyset$$

mentre

$$L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L$$

Dalla definizione di potenza di un linguaggio possiamo ricavare la possibilità di definire un secondo linguaggio che contiene tutte le stringhe che hanno lunghezza minore di un determinato numero  $k$  che è la potenza del linguaggio. Ad esempio:

$$L = \{\varepsilon, a, b\}^3 \quad k = 3$$

$$L = \{\varepsilon, a, b, aa, ab, bb, aaa, \dots, bbb\}$$

L'utilizzo della stringa nulla  $\varepsilon$  ci permette di esprimere le stringhe di lunghezza 1 e 2.

Oltre alle normali operazioni definiamo anche le operazioni insiemistiche sui linguaggi. Tali operazioni sono l'*unione* ( $\cup$ ), l'*intersezione* ( $\cap$ ), la *differenza* ( $\setminus$ ), l'*inclusione* ( $\subseteq$ ), l'*inclusione stretta* ( $\subset$ ) e l'*uguaglianza* ( $=$ ).

Dopo aver introdotto queste operazioni possiamo definire il *linguaggio universale* come insieme di tutte le stringhe di alfabeto  $\Sigma$

$$L_{\text{universale}} = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

$$L_{\text{universale}} = \neg \emptyset$$

Il *complemento* di un linguaggio  $L$  di alfabeto  $\Sigma$  è definito come la differenza insiemistica tra il *linguaggio universale* e il linguaggio  $L$  ovvero:

$$\neg L = L_{\text{universale}} \setminus L$$

ed indica l'insieme delle stringhe di alfabeto  $\Sigma$  e che non appartengono a  $L$ .

Sia nel linguaggio naturale che in quello artificiale le stringhe possono avere una qualsiasi lunghezza, tuttavia per definire un linguaggio è necessario utilizzare formule di lunghezza finita. Risulta perciò necessario introdurre alcuni operatori per creare delle stringhe infinite. L'operatore *star* noto anche come *stella di Kleene* indica la chiusura *riflessiva* e *transitiva* rispetto al concatenamento. Questo operatore indica l'unione di tutte le potenze del linguaggio e si indica come:

$$L^* = \bigcup_{h=0 \dots \infty} L^h = L^0 \cup L^1 \cup L^2 \dots = \varepsilon \cup L^1 \cup L^2 \dots$$

Un esempio di un linguaggio finito è  $L = \{ab, ba\}$  che diventa infinito una volta applicato l'operatore stella:

$$L^* = \{\varepsilon, ab, ba, abba, abab, baba, baab, \dots\}$$

Tuttavia è anche possibile che il linguaggio iniziale  $L$  ed il linguaggio  $L^*$  coincidano.

$$L = \{a^{2n} | n \geq 0\} \quad L^* = \{a^{2n} | n \geq 0\} \equiv L$$

Ogni stringa del linguaggio  $L^*$  può essere suddivisa in una sotto-stringa del linguaggio  $L$ . Se prendiamo un alfabeto  $\Sigma$  come base di un linguaggio,  $\Sigma^*$  contiene tutte le stringhe ( $\Sigma^*$  è il linguaggio universale). Si può anche dire che  $L$  è un linguaggio costruito sull'alfabeto  $\Sigma$  tramite la notazione  $L \subseteq \Sigma^*$ . L'operatore *stella* possiede alcune proprietà importanti:

- monotonicità

$$L \subseteq L^*$$

- chiusura rispetto al concatenamento

$$se \quad (x \in L^* \wedge y \in L^*) \Rightarrow xy \in L^*$$

- idempotenza

$$(L^*)^* = L^*$$

- commutatività di stella e riflessione

$$(L^R)^* = (L^*)^R$$

Inoltre applicando l'operazione stella al linguaggio vuoto e alla stringa nulla otteniamo che:

$$\emptyset^* = \{\varepsilon\} \quad \{\varepsilon\}^* = \{\varepsilon\}$$

Un esempio di idempotenza è già stato mostrato ed era:

$$L = \{a^{2n} | n \geq 0\} \quad L^* = \{a^{2n} | n \geq 0\} \equiv L$$

questo perché il linguaggio  $L$  può essere visto come:

$$L = L_0^* \quad \text{dove } L_0 = \{aa\}$$

L'operatore *croce* è la *chiusura transitiva* rispetto al concatenamento, ovvero è l'unione delle potenze tranne la potenza 0; in molti casi è molto utile tuttavia non è indispensabile in quanto deriva dall'operatore stella.

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

Operatore *quoziente* (/) accorcia una frase di un primo linguaggio eliminando un suffisso appartenente ad un secondo linguaggio.

*Notare che a differenza dell'operatore differenza la barra in questo caso è rivolta in avanti*

$$L = L' / L'' = \{y | (x = yz \in L') \wedge z \in L''\}$$

Un esempio di questo utilizzo è:

$$\begin{aligned} L' &= \{a^{2n}b^{2n} | n > 0\}, \quad L'' = \{b^{2n+1} | n \geq 0\} \\ L' / L'' &= \{a^r b^s | (r \geq, \text{pari}) \wedge (1 \leq s < r, s \text{ dispari})\} \\ &= \{a^2b, a^4b, a^4b^3, \dots\} \end{aligned}$$

regexp	Linguaggio
$\varepsilon$	$\{\varepsilon\}$
$a \in \Sigma$	$\{a\}$
$s \cup t$	$L_s \cup L_t$
$s \cdot t$	$L_s \cdot L_t$
$s^*$	$L_s^*$

Tabella 1: Regole di generazione dei linguaggi da regexp

## 2 Linguaggi ed espressioni regolari

I *linguaggi regolari* sono la famiglia più semplice di linguaggi formali, questo perchè possono essere definiti in diversi modi:

- Algebricamente.
- Attraverso grammatiche generative.
- Attraverso algoritmi di riconoscimento (*automi*)

Le *espressioni regolari* (*regexp*) sono stringhe  $r$  definite attraverso i caratteri di un alfabeto  $\Sigma$  e mediante l'utilizzo dei metasimboli  $\emptyset, \cup, \cdot, *$  attraverso l'applicazione e la combinazione delle seguenti regole

- $r = \emptyset$
- $r = a, a \in \Sigma$
- $r = (s \cup t)$
- $r = (s \cdot t)$  o  $r = (st)$
- $r = (s)^*$

Oltre a questi simboli talvolta, per semplificare la notazione, si utilizzano anche i simboli  $\varepsilon = \emptyset^*$  e  $r^+ = r \cdot r^*$ .

Il risultato di un'espressione regolare  $e$  è un linguaggio  $L_e$  costruito sull'alfabeto  $\Sigma$  che rispecchia le regole di Tabella 1. A questo punto possiamo dire che un linguaggio è *regolare* se è generato tramite l'utilizzo di *espressioni regolari* come ad esempio

$$e = (111)^* \quad L_e = \{1^{3n} | n = 0, 1, \dots\}$$

### 2.1 La famiglia dei linguaggi regolari

Definiamo la *famiglia dei linguaggi regolari* (REG) come la collezione di tutti i linguaggi regolari, inoltre, definiamo la *famiglia dei linguaggi finiti* (FIN) la collezione di tutti i linguaggi aventi *cardinalità finita*.

Ogni linguaggio finito è regolare in quanto è l'unione ( $\cup$ ) di un numero finito di stringhe le quali sono il concatenamento ( $\cdot$ ) di un numero finito di caratteri. Tuttavia l'insieme dei linguaggi regolari contiene anche linguaggi non finiti perciò l'inclusione risulta essere stretta.

$$FIN \subset REG$$



$(a_1 \cup (b_2 b_3))^*$	$c_4^+ \cup (a_5 \cup (b_6 b_7))$
$a_1 \cup (b_2 b_3)$	$c_4^+ \quad a_5 \cup (b_6 b_7)$
$a_1 \quad b_2 b_3$	$c_4 \quad a_5 \quad b_6 b_7$
$\quad b_2 \quad b_3$	$\quad \quad b_6 \quad b_7$

Figura 4: Scomposizione in sottoespressioni

## 2.2 Sottoespressione di un'espressione regolare

Consideriamo un espressione completamente parentizzata come la seguente:

$$e = (a \cup (bb))^*(c^+ \cup (a \cup (bb)))$$

A questo punto distinguiamo tutti i termini numerandoli

$$e = (a_1 \cup (b_2 b_3))^*(c_4^+ \cup (a_5 \cup (b_6 b_7)))$$

Infine mettiamo in evidenza tutte le sotto espressioni come mostrato in Figura4

## Elenco delle figure

1	Esempio di sottostringhe con prefissi e suffissi . . . . .	4
2	Proprietà della riflessione . . . . .	5
3	Esempio di ripetizione . . . . .	5
4	Scomposizione in sottoespressioni . . . . .	9