LC - Linguaggi e Computabilità

Elia Ronchetti
@ulerich

Settembre 2022

Indice

Capitolo 1

Linguaggi formali

Preso un alfabeto $\Sigma = \{a, b, c\}$ posso formare delle stringhe come $\{bab, aca, b\}$, questo insieme è un linguaggio sull'alfabeto Σ.
Con un linguaggio posso fare principalmente 2 cose

- Riconoscitori - detti anche **automi**, sono degli automi che prendono in input una stringa e mi dicono se appartiene o no al linguaggio, **riconoscono** quindi il linguaggio
- Generatori - sono le Grammatiche, dalla grammatica posso generare il linguaggio, esse mi danno diverse regole attraverso le quali posso generare un linguaggio.

Alfabeto Un alfabeto è un insieme non vuoto e finito di simboli

DEFINIZIONE

Un alfabeto è un insieme non vuoto e finito di simboli definito con la lettera Σ

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

DEFINIZIONE

Una **stringa** è una sequenza finita di simboli dell'alfabeto.
In questo caso è ammesso l'insieme vuoto, si parla di stringa vuota e viene definita come ϵ

$$L \subseteq \Sigma^*$$

La lunghezza di una stringa viene denotata dai seguenti simboli |stringa|.

Esempio: $|0111| = 4$

Σ^n è una stringa di lunghezza.

DEFINIZIONE

Un **Linguaggio** (su un alfabeto Σ è un sottoinsieme di Σ^*)

Notazioni particolari

- $\Sigma^0 = \{\epsilon\}$
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup ...$ oppure $\Sigma^* = \Sigma^+ \cup \Sigma^0 = \Sigma^+ \cup \{\epsilon\}$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ...$ oppure $\Sigma^+ = \Sigma^* - \{\epsilon\}$

Notazioni stringhe Per definire una stringa utiizzo una lettera come per esempio $w = abb$ dove in w sono contenuti dei simboli dell'alfabeto $\Sigma = \{a, b, c\}$.
Definita un'altra stringa $y = cca$ definisco la concatenazione di due stringhe come $wy = abbcca$. La concatenazione **NON** è commutativa.

Capitolo 2

Le Grammatiche

Una grammatica è una quadrupla

$$G = (V, T, P, S)$$

Dove

- V è l'insieme delle Variabili
- T è l'insieme dei simboli Terminali
- P è l'insieme delle regole di Produzione
- S $\in$ V, è lo Start symbol

DEFINIZIONE

Grammatica: termine che designa una struttura formale per un linguaggio L in grado di generare tutte e sole le stringhe del linguaggio. Per questo si parla di grammatica G generativa del linguaggio L o di linguaggio L generato dalla grammatica G, indicandolo con L(G).

Definizione Treccani Link alla definizione

2.1 Gerarchia di Chomsky

La gerarchia di Chomsky è un insieme di classi di grammatiche formali che generano linguaggi formali. Suddivide le grammatiche in 4 tipi:

- Tipo 0 - Linguaggi ricorsivamente enumerabili

- Tipo 1 - Linguaggi Contestuali
- Tipo 2 - Linguaggi Context-Free (Liberi dal contesto)
- Tipo 3 - Linguaggi Regolari

2.1.1 Tipo 0

Sono definiti nel seguente modo:

$$\alpha \to \beta, \text{ con } \alpha \text{ e } \beta \in (V \cup T)^*$$

Esempio: $0S1S0 \to 00SS1S01$

Sono detti **ricorsivamente enumerabili** e vengono accettati dalle **Macchine di Turing deterministiche e non deterministiche**.

2.1.2 Tipo 1

Verranno solo visti un paio di esempi, ma non saranno trattati in questo corso.
Definiti nel seguente modo:

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$$
$$A \in V \qquad \alpha_1, \alpha_2, \beta \in (V \cup T)^*$$

In questo linguaggio sono presenti più vincoli rispetto al tipo 0. Questi linguaggi sono detti contestuali, questo perchè posso sostituire le variabili solo in un determinato contesto, in questo caso solo se sono presenti sia α_1 e α_2 tra β.
Sono accettati dalle **Macchine di Turing con nastro "lineare"**.

2.1.3 Tipo 2

$A \to \gamma \qquad A \in V, \gamma \in (V \cup T)^*$.
Detti anche **Context-Free (Liberi dal Contesto)**.
Riconosciuti da **Automi a Pila non deterministici**

2.1.4 Tipo 3

$A \to aB$ oppure $A \to a$ oppure
$A \to Ba$ oppure $A \to a$.
Sono detti **Regolari** e sono riconosciuti da **Automi a stati finiti deterministici e non deterministici**.

Osservazione Scorrendo verso il basso la gerarchia noto che le grammatiche sono sempre più stringenti e meno libere.

Produzione di un linguaggio

Per produrre un linguaggio è necessario definire la relativa grammatica (V, T, P, S). Quando scrivo le regole di produzione non posso imporre un ordine di applicazione delle regole e non posso negare l'applicazione di una regola, tutte le regole possono essere applicate in qualsiasi ordine.

Regole di Produzione Con regole di produzione si intendono le regole che mi permettono di passare da una variabile ad un simbolo per poter costruire una frase.

Esempio: Bilanciamento parentesi
$G_bal = \{\{S\}, \{(,)\}, P, S\}$
P$\{S \to SS|(S)|\epsilon\}$
In questo caso la freccia indica una regola di produzione
Esempi di frasi valide $\to$ $(), (()), ()()$
Esempi di frasi non appartenentei al linguaggio $\to)($
Applicazione regole di produzione
$S \Rightarrow \underline{S}S \Rightarrow (S)\underline{S} \Rightarrow (S)(\underline{S}) \Rightarrow (\underline{S})() \Rightarrow ((\underline{S}))() \Rightarrow (())()$
Sostituendo i le variabili attraverso le regole di produzione ottengo le frasi appartenenti al linguaggio.
In questo caso $\Rightarrow$ indica un passaggio di derivazione

Attenzione Non si effettuano mai più sostituzioni in un unico passaggio, dato che non è garantito che il risultato ottenuto sia corretto.
Con $\Rightarrow^*$ indico che faccio uno o più passi di derivazione (NON più sostituzioni in un unico passaggio).

2.2 Context Free Grammar (CFG) - Linguaggi liberi dal contesto (Tipo 2)

Questi linguaggio sono caratterizzati da una definizione ricorsiva dello stesso, qui di seguito alcuni esempi per chiarire cosa si intende.

Esempio: *Fornire una CFG per il linguaggio* $L = \{0^n 1^n | n \geq 1\} n \in \mathbb{N}$
Alcuni esempi di L sono = {01, 0011, 000111, ...}
Se $winL$ allora $0w1 \in L$
$S \to 01|0S1$
$V = \{S\}$, $T = \{0, 1\}$ $G = (V, T, P, S)$
$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$
n ci indica quante volte il simbolo si deve ripetere come minimo, in questo caso per esempio essendo ≥ 1 *ci sta dicendo che vuole che il simbolo sia ripetuto almeno 1 volta, quindi* ϵ *non è ammesso, fosse stato* ≥ 0 *significava che* ϵ *appartiene al linguaggio*

Left Most e Right Most derivation

Posso applicare le regole di derivazione in diversi ordini, per esempio applicando sempre la variabile più a sinistra o più a destra, vedremo che questo non cambia il risultato, disegnando un albero sintattico noteremo come i due rami a seconda del tipo di applicazione risulteranno invertiti, ma il risultato sarà sempre lo stesso.

Risoluzione esercizi

Lo svolgimento degli esercizi è sui PDF presenti nella cartella drive indicata nel README.md, comunque qui di seguito sono elencati consigli per la risoluzione.

Incroci Nelle CFG non possono esserci "incroci" fra esponenti uguali, come per esempio

Esempio: $a^n b^m c^n y^m$

In questo caso le n e m risultano incrociate, non posso suddividerle in pattern o sottopattern e questo mi impedisce di produrre una CFG.

- Ragionare sul pattern delle lettere e non sulla singola lettera
- Partire dall'esterno e mano a mano passare ai pattern più interni per creare le regole di produzione
- Per ogni pattern identificato sarà necessario utilizzare una Variabile
- Quando ho situzioni di questo tipo $a^{n+m}xc^nyd^m$ è necessario prestare attenzione a come si dividono i due esponenti perchè a volte posso generare degli incroci, per questo è necessario scomporre l'esponente nell'ordine corretto, quello che mi permette di ottonere gruppi non incrociati
- La situazione precedente tipicamente si presenta negli esercizi dove come condizioni ho $n \geq m \geq 0$

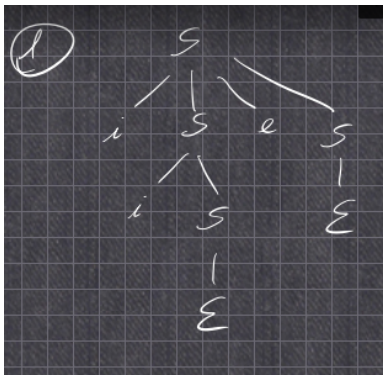
2.3 Linguaggi di Tipo 1 (Contestuali)

Le Grammatiche di Tipo 1 sono dipendenti dal contesto, questo significa che durante la produzione di una stringa viene sostituito un simoblo alla volta, ma solo quando è in presenza di un altro simbolo. Vedremo che nella testa delle regole di produzione avrò la concatenazione di due variabili. Nell'esempio riportato a questo **link** vengono mostrate le regole di produzioni necessarie per generare un esempio di linguaggio di tipo 1.

2.4 La Grammatiche Ambigue

2.4.1 Albero Sintattico

Data una CFG $G = (V, T, P, S)$ un modo per rappresentare le regole di produzione applicate è rappresentare l'albero sintattico. Esso consiste in un alberto dove le foglie sono etichettate con una variabile, da un simbolo terminale o da ϵ. Ogni nodo interno è etichettato con una variabile, se è etichettato da ϵ è l'unica foglia del suo genitore.
Qui di seguito un esempio di albero per la derivazione della stringa *iee*



Se un nodo interno è etichettato con A e i figli sono etichettati da sinistra con $X_1, X_2, ..., X_n$ allora $A \rightarrow X_1, X_2, X_k \in P$. La definizione è importante per l'argomento che adesso andremo a trattare

Ambiguità

Se una grammatica consente la generazione della stessa stringa attraverso due alberi sintattici diversi allora quella è una grammatica ambigua.

DEFINIZIONE

1. $\exists w \in T^*$ tale che w ha due alberi sintattici diversi *rightarrow* Grammatica Ambigua
2. $\forall w \in T^*$ w ha un solo albero sintattico $\rightarrow$ Grammatica NON Ambigua

NB Ottenere la stessa stringa con derivazioni diverse non è un problema, il problema è quando ottengo **Alberi sintattici diversi!**. Se una Grammatica è ambigua posso risolvere il problema e renderla non ambigua, mentre se un linguaggio è ambiguo NON posso risolvere il problema dato che un linguaggio ambiguo può essere generato solo da Grammatiche ambigue.

Problema Non esiste un algoritmo in grado di identificare una grammatica ambigua. Gli esempi pratici li trovate nelle ultime pagine di questo PDF - **Lezione 4**.

2.5 Grammatiche Regolari (Tipo 3)

1. ϵ può comparire solo in $S \rightarrow \epsilon$, dove S è lo start symbol

2. le regole di produzione sono tutte lineari a destra oppure tutte lineari a sinistra

Esempio: Lineare a Destra
$A \to aB$
$A \to a$ dove $A, B \in V \ a \in T$

Esempio: Lineare a Sinistra
$A \to Ba$
$A \to a$ dove $A, B \in V \ a \in T$

Non si possono mischiare questi due esempi, o sono lineare a sinistra o a destra.

Capitolo 3

Espressioni Regolari

DEFINIZIONE

Un'espressione regolare (in lingua inglese regular expression o, in forma abbreviata, regexp, regex o RE) è una sequenza di simboli (quindi una stringa) che identifica un insieme di stringhe. Possono definire tutti e soli i linguaggi regolari.

3.1 Operazioni Tra Linguaggi

Unione

Dati $L, M \rightarrow L \cup M$

Esempio: L = {001, 10,111}
M = { ϵ , 001}
L ∪ M = { ϵ , 001, 10, 111}

Gli elementi in comune non si ripetono

Concatenazione

$L, M \rightarrow LM$

Esempio: L = {001, 10, 111}
M = { ϵ , 001}

LM = {001, 001001, 10, 10001, 111, 111001}

Tutte le possibili concatenazioni

Chiusura di Kleene

$L = \emptyset$

Esempio: $\emptyset^0 = \{\epsilon\}$
$L^0 = \{\epsilon\} \forall L$
$\emptyset^i = \emptyset \forall_i \geq 1$
$\emptyset^* = \emptyset^0 \cup \emptyset^1 \cup \emptyset^2 \cup \cdots =$
$\{\epsilon\} \cup \emptyset \cup \emptyset \cup \cdots =$
$\{\epsilon\}$

Espressioni Regolari

NB In questo contesto ϵ e $\emptyset$ sono ER (Espressioni Regolari) NON sono stringhe.

1. ϵ e $\emptyset$ sono ER $\rightarrow$ L(ϵ) = { ϵ } L($\emptyset$) = $\emptyset$
2. Se $a \in \Sigma$ allora a è una ER (Dove Σ è alfabeto)
3. Variabili che rappresentano linguaggi sono ER

Induzione

1. **Unione** $\rightarrow$ se E,F sono ER allora E+F è una ER
   $L(E + F) = L(E) \cup L(F)$
2. **Concatenazione** $\rightarrow$ se E,F sono ER allora EF è una ER
   $L(EF) = L(E) * L(F)$
3. **Chiusura** $\rightarrow$ Se E è ima ER allora E* è una ER
   $L(E^*) = (L(F))^*$
4. **Parentesi** $\rightarrow$ Se E è una ER allora (E) è una ER
   $L((E)) = L(E)$

Proprietà

- **Unione** $\rightarrow$ Commutativa, Associativa
  $L + M = M + L$
  $(L + M) + N = L + (M + N) = L + M + N$
- **Concatenazione** $\rightarrow$ Associativa, NON è commutativa
  $(LM)N = L(MN)$
  $01 \neq 10$