Appunti di Linguaggi e Computabilità

Linguaggi e Computabilità (prof. Pomello) - CdL Informatica Unimib - 24/25

Federico Zotti

20 ottobre 2024

Indice

1 Definizioni

> Questa prima sezione è stata fatta con il prof. del T1.

Alfabeto: insieme finito e non vuoto di simboli. Es:

$$\Sigma = \{ 0, 1 \} \qquad \Sigma = \{ a, b, c, \dots, z \}$$

Stringa: sequenza finita di simboli appartenenti ad un alfabeto. Es:

$$w = 01101 \qquad s = abca$$

Stringa vuota: sequenza che non contiene nessun simbolo (ε, λ). Attenzione che $\varepsilon \notin \Sigma,\ \forall \Sigma$.

Lunghezza di una stringa: numero di simboli (posizioni) di una stringa. Es:

$$w = 01101 \qquad |w| = 5$$
$$|\varepsilon| = 0$$

1.1 Potenze di un alfabeto

Dato un alfabeto Σ esiste la potenza Σ^k con $k \geq 0$ intero.

> **Es:**
>
> $$\Sigma = \{ a, b \}$$
> $$\Sigma^2 = \{ aa, ab, ba, bb \}$$
> $$\Sigma^0 = \{ \varepsilon \} \neq \varnothing$$
>
> **Attenzione:** $\Sigma \neq \Sigma^1$, perché Σ è un insieme di simboli, mentre Σ^1 è un insieme di stringhe.

$$|\Sigma| = n \implies |\Sigma^k| = n^k$$

1.2 Σ^* e Σ^+

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots = \bigcup_{i \geq 0} \Sigma^i$$

Es:

$$\Sigma = \{ 0, 1 \}$$
$$\Sigma^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots \}$$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots = \bigcup_{i \geq 1} \Sigma^i$$

dunque

$$\Sigma^+ = \Sigma^* \setminus \{ \varepsilon \}$$
$$\Sigma^* = \Sigma^+ \cup \{ \varepsilon \}$$

1.3 Concatenazione di stringhe

Es:

$$u = 010 \qquad v = 1101$$
$$u \circ v = u \cdot v = uv = 0101101$$
$$|uv| = |u| + |v|$$
$$u\varepsilon = u = \varepsilon u$$

ε è l'elemento neutro di questa operazione (identità).

La concatenazione non è commutativa ($xy \neq yx$), mentre è associativa ($x(yz) = (xy)z$).

Dunque si può avere il **monoide** su Σ non commutativo

$$\langle \Sigma^*, \cdot, \varepsilon \rangle$$

1.4 Linguaggio

Un linguaggio $L \subseteq \Sigma^*$ è un sottoinsieme di stringhe.

> **Es:**
> Dati un grafo orientato G, due nodi s, t, stabilire se esiste in G un cammino da s a t.
> Questo è un problema di decisione.
> Si può anche scrivere come:
>
> - $\Sigma = \{ 0, 1 \}$.
> - Σ^* Sono le stringhe utilizzate per rappresentare G (i nodi e gli archi).
> - $L = \{ w \in \Sigma^* \mid w \text{ rappresenta } G, s, t \text{ e esiste un cammino da } s \text{ a } t \}$.
>
> $$w \in \Sigma^* \to A_L \to \text{Si se } w \in L \quad \vee \quad \text{No se } w \notin L$$
>
> Questo è un problema di membership.

1.4.1 Cardinalità

$$card(\Sigma^*) \sim \mathbb{N}$$
$$card(\mathcal{L}) \sim \mathbb{R}$$

con $\mathcal{L}$ l'insieme di tutti i possibili linguaggi su Σ.

Quindi la cardinalità di un linguaggio L può essere sia finita che infinità, ma con cardinalità massima $\mathbb{N}$.

2 Linguaggi formali

2.1 Definizione formale di lunghezza

Definizione per induzione di lunghezza di una stringa:

$$|\cdot| : \Sigma^* \to \mathbb{N}$$

> Σ^* insieme di tutte le stringhe.

Passo base:

$$|\varepsilon| = 0 \quad ; \quad |a| = 1 \ \forall a \in \Sigma$$

Passo induttivo:

Se $w = ax$ e conosco $|x|, x \in \Sigma^*$, allora

$$|w| = |ax| = 1 + |x|$$

2.2 Rappresentazione di un linguaggio

Il nostro problema è rappresentare un linguaggio ($L \subseteq \Sigma^*$) in maniera finita.

Un linguaggio può essere classificato in due modi:

- A seconda di come genero le stringhe di L (**Grammatica**)
- A seconda di come riconosco le stringhe di L (**Automi**)

3 Grammatiche

Le grammatiche si dividono in 4 categorie:

- **Tipo 3:** Espressioni regolari (automi a stati finiti)

- **Tipo 2:** Grammatiche *context-free* (automi a pila)
- **Tipo 1:** Grammatiche dipendenti dal contesto (automi limitati lineari)
- **Tipo 0:** Linguaggi ricorsivamente enumerabili (macchine di Turing)

$$\text{Tipo 3} \subset \text{Tipo 2} \subset \text{Tipo 1} \subset \text{Tipo 0}$$

o meglio

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$$

3.1 Grammatiche di tipo 2

Le grammatiche di tipo 2, o **context-free**, sono alla base delle sintassi dei linguaggi di programmazione nella *Backus-Naur Form* (BNF).

3.1.1 Esempio: Linguaggio delle stringhe palindrome

Definiamo L_{Pal} su $\Sigma = \{ 0, 1 \}$ per induzione.

Passo base:

$$\varepsilon \in L_{\text{Pal}} \qquad 0 \in L_{\text{Pal}} \qquad 1 \in L_{\text{Pal}}$$

Passo induttivo:

Se $w \in L_{\text{Pal}}$ allora

$$0w0 \in L_{\text{Pal}} \qquad 1w1 \in L_{\text{Pal}}$$

Nessuna altra stringa in $\Sigma^* \in L_{\text{Pal}}$.

Regole di produzione:

$$
\begin{array}{ll}
1 & P \to \varepsilon \\
2 & P \to 0 \\
3 & P \to 1 \\
4 & P \to 0P0 \\
5 & P \to 1P1
\end{array}
$$

Questo ci permette di definire una grammatica

$$G_{\text{Pal}} = \left(T = \{\, 0, 1 \,\},\ V = P,\ \mathcal{P} = \{\, {}_1, {}_2, {}_3, {}_4, {}_5 \,\},\ S = P\right)$$

3.2 Grammatiche di tipo 0

Una grammatica è così definita:

$$G = \left(V, T, \mathcal{P}, S\right)$$

con:

- V: insieme finito di simboli **variabili**
- T: ins. finito di simboli **terminali**
- $S \in V$: variabile di **start**
- $\mathcal{P} = \{\, \alpha \to \beta \mid \alpha \in (V \cup T)^{+} \wedge \alpha$ contiene almeno una variabile $\wedge\, \beta \in (V \cup T)^{*} \,\}$: **produzioni**

> Esempio produzione di tipo 0:
>
> $$1A0 \to 1B01$$

Nel caso di produzioni di tipo 2

$$\mathcal{P} = \{\, x \to \beta \mid x \in V \wedge \beta \in (V \cup T)^{*} \,\}$$

> La differenza è che nelle grammatiche di tipo 0 è consentito utilizzare simboli oltre alla variabile a sinistra della produzione.

3.3 Derivazioni

Def. stringhe generate da $G = \{ V, T, \mathcal{P}, S \}$.

Def: $\underset{G}{\Rightarrow}$ è definita come:

Sia $\alpha A \beta \in (V \cup T)^+$

Se $A \to \gamma \in \mathcal{P}_G \quad \gamma \in (V \cup T)^*$ allora $\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$

Def: La derivazione $\underset{G}{\Rightarrow}^*$ è invece la chiusura riflessiva e transitiva di $\underset{G}{\Rightarrow}$.

Base: $\forall \alpha \in (V \cup T)^+ \quad \alpha \underset{G}{\Rightarrow}^* \alpha$

Ipotesi:

Se $\alpha \underset{G}{\Rightarrow}^* \beta \wedge \beta \underset{G}{\Rightarrow} \gamma$ allora $\alpha \underset{G}{\Rightarrow}^* \gamma$.

Es:

$$P \underset{G}{\Rightarrow} 1P1 \underset{G}{\Rightarrow} 10P01 \underset{G}{\Rightarrow} 10001$$
$$P \underset{G}{\Rightarrow}^* 10001$$

Dunque il linguaggio generato da G è

$$L(G) = L_G = \{ w \in T^* \mid S \underset{G}{\Rightarrow}^* w \}$$

Es:

$$L = \left\{ w \in \{ 0, 1 \}^* \mid w = 0^n 1^n \wedge n \geq 0 \right\}$$

Definiamo quindi la grammatica

$$G = \left(T = \{ 0, 1 \}, V = \{ S \}, \mathcal{P}, S\right)$$

Con le produzioni $\mathcal{P}$:

$$_1S \to \varepsilon$$
$$_2S \to 0S1$$

In questo caso $S \to 01$ è ridondante, perché può essere generato da $_2$.

Per un linguaggio invece

$$L' = \left\{w \in \{ 0, 1 \}^* \mid w = 0^m 1^m \wedge m > 0\right\}$$

Si avranno delle produzioni:

$$_1S' \to 01$$
$$_2S' \to 0S'1$$

3.4 Esempio: ling. delle parentesi bilanciate

$$G = \left\{V = \{ B \}, T = \{ (,) \}, \mathcal{P}, S = B\right\}$$

B sono le parentesi bilanciate.

Le produzioni sono:

$$_1B \to \varepsilon$$
$$_2B \to (B)$$

Queste si possono anche scrivere come

$$B \to \varepsilon \mid (B)$$

Quindi per esempio si possono generare le seguenti stringhe:

$$B \Rightarrow (B) \Rightarrow ((B)) \Rightarrow ((\,))$$

$$B \Rightarrow^* ((\ldots(\,)\ldots))$$

Come posso però generare stringhe come $((\ldots(\ldots)\ldots(\ldots)\ldots))$?

Dobbiamo modificare le produzioni:

$$B \to \varepsilon \mid (B) \mid BB$$

A seconda se si sostituisce prima la variabile più a sinistra o più a destra si può dire che la derivazione è *right-most* o *left-most*.

Teo:

$$G = \{V = \{\ldots\}, T = \{(,)\}, \mathcal{P}, S\}$$

Per le grammatiche di tipo 2:

$$\forall\ A \in V \quad \forall\ w \in T^*$$

$$A \underset{G}{\Rightarrow^*} w \iff A \underset{\text{lm}}{\Rightarrow^*} w \iff A \underset{\text{rm}}{\Rightarrow^*} w$$

Per ogni derivazione esiste una derivazione right-most e left-most.

3.5 Albero sintattico

È possibile creare un albero che rappresenta tutte le derivazioni applicate ad una stringa.

$$B \underset{\text{rm}}{\Rightarrow^*} (()())$$

```
    B
 /  |  \
```

```
(     B     )
     / \
    B    B
   /|\   /|\
  ( B )( B )
    |     |
   eps   eps
```

> #todo-uni Rifare l'albero

La frontiera di questo albero è la concatenazione delle foglie dell'albero, e essendo ε l'elemento neutro della concatenazione, si ottiene

$$(()())$$

Def: albero sintattico (o di derivazione).

$$G = (V, T, \mathcal{P}, S)$$

Se $w \in L(G)$, allora w è la frontiera di un albero sintattico.

Un albero è un albero sintattico per G sse

- Ogni nodo interno è etichettato da $x \in V$
- Ogni foglia è etichettata da $x \in V \vee a \in T \vee \varepsilon$ (nel caso di ε, il nodo è l'unico figlio del padre)
- Se un nodo x ha k figli etichettati $y_1 \dots y_k$ ed è un sottoalbero, allora si ha una produzione $x \to y_1 \dots y_k \in \mathcal{P}_G$ $(x \in V; y_i \in V \cup T)$
- Se un nodo x ha solo un figlio ε, allora $x \to \varepsilon \in \mathcal{P}_G$

$$L(G) = \{\ w \in T^* \mid w \text{ è frontiera di un albero sint. con radice } S\ \}$$

Data $w \in L(G)$, esiste un albero sintattico con frontiera w.

$$\exists ! S \underset{\text{rm}}{\Rightarrow^*} w \quad \wedge \quad \exists ! S \underset{\text{lm}}{\Rightarrow^*} w$$

3.6 Esempi grammatiche di tipo 2

$$L = \left\{ w \in \{ a, b, c, d \}^* \mid w = a^m b^m c^n d^n, m \geq 0, n > 0 \right\}$$

$$w = \underbrace{a \dots a}_{m} \underbrace{b \dots b}_{m} \underbrace{c \dots c}_{n} \underbrace{d \dots d}_{n}$$

$$\mathcal{P} : \quad \begin{aligned} & S \to XY \\ & X \to \varepsilon \mid aXb \\ & Y \to cd \mid cYd \end{aligned}$$

3.7 INSERIRE LEZIONE 10/10/2024!

> Grammatiche ambigue (?)

3.8 Grammatiche ambigue

Grammatica G, produzioni $\mathcal{P}_G$:

$$\begin{aligned} & E \to I \mid E + E \mid E * E \mid (E) \\ & I \to a \mid b \end{aligned}$$

È possibile creare due alberi differenti per la stringa "$a + b * a$":

- $E \Rightarrow E + E \Rightarrow E + E * E$
- $E \Rightarrow E * E \Rightarrow E + E * E$

Quindi G è una grammatica ambigua. È possibile creare una grammatica G' che produca le stesse stringhe ma che non sia ambigua.

$$L(G) = L(G') \quad \wedge \quad G' \text{ non ambigua}$$

$$\mathcal{P}_{G'} : \quad \begin{aligned} E &\to T \mid E + T \\ T &\to F \mid T * F \\ F &\to I \mid (E) \\ I &\to a \mid b \end{aligned}$$

Quindi "$a + b * a$" è esprimibile solo con la seguente derivazione left-most:

$$\begin{aligned} E &\Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow I + T \\ &\Rightarrow a + T \Rightarrow a + T * F \Rightarrow a + F * F \\ &\Rightarrow a + I * F \Rightarrow a + b * F \Rightarrow a + b * I \\ &\Rightarrow a + b * a \end{aligned}$$

Se $\exists G$ non ambigua t.c. $L = L(G)$, allora L non è ambiguo.

L è **inerentemente ambiguo** se $\forall G$ t.c. $L(G) = L$, G è ambigua.

Esempio:

$$L = L_1 \cup L_2$$

dove $L1 \cap L_2 \neq \varnothing$.

$$L = \left\{ w \in \{ a, b, c, d \}^* \mid w = a^m b^m c^n d^n \ \vee \ w = a^m b^n c^n d^m, \ n, m > 0 \right\}$$

$$S \to XY \mid Z$$
$$X \to aXb \mid ab$$
$$Y \to cYd \mid cd$$
$$Z \to aZd \mid aVd$$
$$V \to bVc \mid bc$$

Se prendiamo per esempio la stringa "$aabbccdd$", si può generare con la derivazione left-most:

$$S \Rightarrow XY \Rightarrow aXbY \Rightarrow aabbY \Rightarrow aabbcYd$$
$$\Rightarrow aabbccdd$$

e con la derivazione left-most:

$$S \Rightarrow aZd \Rightarrow aaVdd \Rightarrow aabVcdd \Rightarrow aabbccdd$$

3.9 Grammatiche di tipo 3

Permettono di generare linguaggi di tipo 3 o **linguaggi regolari**.

Le produzioni $\mathcal{P}$ di tipo 3 hanno solo una variabile a sinistra dell'operatore.

Se il linguaggio include ε, allora deve avere questa produzione:

$$S \to \varepsilon$$

Inoltre *tutte* le produzioni sono uno di questi tipi (un solo tipo per grammatica):

- **lineari a destra**: $A \to aB$ oppure $A \to a$
- **lineari a sinistra**: $A \to Ba$ oppure $A \to a$

$$a \in T \wedge A, B \in V$$

Se G è lin. sin., allora $\exists G'$ lin. des. t.c. $L(G) = L(G')$. E chiaramente viceversa.