LC - Linguaggi e Computabilità

Elia Ronchetti
@ulerich

2022/2023

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.
Un linguaggio può essere generato o riconosciuto:

- 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:

$$\begin{gathered} \alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2 \\ A \in V \qquad \alpha_1, \alpha_2, \beta \in (V \cup T)^* \end{gathered}$$

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 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.

2.3 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,h 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*

2.3.1 Definizione formale

Una grammatica di tipo Context-Free è definita come una quadrupla del tipo:

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

- V - Insieme delle Variabili
- T - Insieme dei simboli Terminali
- P - Insieme delle regole di Produzione
- S$\in V$ - Simbolo iniziale o Start Symbol

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

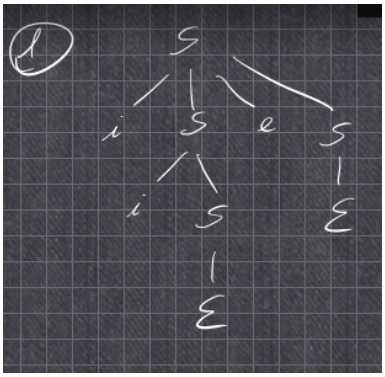
In questo caso le n e m risultano incrociate, non posso suddividerle in pattern o sottopattern e questo mi impedisce di produrre una CFG.

- 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 $a^{n+m} x c^n y d^m$ 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 $n \geq m \geq 0$

2.4 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 $A \to X_1, X_2, X_k \in P$. 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 $\to$ 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 $S \to \epsilon$, 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
  $(LM)N = L(MN)$
  $01 \neq 10$

Capitolo 4

DFA e NFA

4.1 DFA

I DFA (Deterministic Final Automa) o **Automi a Stati Finiti Deterministici** un tipo di automa in grado di accettare grammatiche di tipo 3 (Regolari). É una quintupla definita come segue:

$$A = Q, \Sigma, \delta, q_0, F$$

Dove delta è una funzione che prende come input uno stato q e un simbolo dell'alfabeto Σ e restituisce uno stato.

$$\delta(q_0, a) = q_1$$

Nei DFA delta è una funzione totale, cioè tutte le combinazioni stato stringa sono definite (nei NFA non è così).

4.2 NFA

I NFA (Non Deterministic Final Automa) o **Automi a Stati Finiti Non Deterministici** sono sempre definiti come quintupla definita come segue:

$$A = Q, \Sigma, \delta, q_0, F$$

In questo caso la δ prende sempre un carattere e uno stato come input, ma restituisce un sottoinsieme di Q, quindi un insieme di stati. Questo significa che l'automa può assumere più stati in contemporanea, con un simbolo per esempio posso andare in 2 stati diversi.

Differenza tra NFA e DFA L'unica differenza tra un NFA e un DFA è quindi nel tipo di valore restituito da δ: un insieme di stati nel caso di un NFA e un singolo stato nel caso di un DFA.

NFA e DFA, hanno la stessa potenza? Sì, ogni NFA può essere convertito in un DFA, c'è da tenere in considerazione che gli NFA sono più compatti dei DFA in termini di rappresentazione grafica.

4.3 Epsilon NFA

Un ϵ NFA è come un NFA che però può eseguire delle ϵ-mosse, cioè può andare da un stato a un altro senza consumare un simbolo della stringa data in input. Formalmente è sempre denotato come $A = (Q, \Sigma, \delta, q_0, F)$ dove tutti i componenti si interpretano come nel caso di un NFA, tranne δ che è una funzione che richiede come argomenti:

- Uno stato in Q
- un elemento di $\Sigma \cup \{\epsilon\}$, cioè un simbolo di input o il simbolo ϵ. Si richiede che il simbolo di stringa vuota ϵ non sia un elemento dell'alfabeto Σ (se no si fa confusione)

DEFINIZIONE

Eclose(q)
è l'insieme di stati di Q che possono essere raggiunti a partire da q facendo solo ϵ-mosse, compreso q stesso.

A livello pratico per ϵ-chiudere uno stato q si seguono tutte le ϵ-transizioni uscenti da q, ripetendo poi l'operazione da tutti gli stati raggiunti via via, fino a trovare tulli gli siati raggiungibili da q attraverso cammini etichettati solo da ϵ-transizioni.

4.4 Conversione da NFA a DFA

Per ogni NFA c'è un corrispondente DFA. Per convertire un ϵ-NFA a DFA è necessario costruire una tabella, prendo lo stato iniziale e ne faccio l'eclose, scrivo l'insieme di stati risultante (se presente uno stato finale allora lo segno con un asterisco) e poi per ogni simbolo dell'alfabeto faccio la δ (verifico

in che stati l'automa si troverà dopo aver consumato il carattere) e l'eclose dell'insieme degli stati ottenuto, scrivo l'insieme di stati risultante (ancora una volta se presente uno stato finale allora segno con un asterisco tutto l'insieme). Una volta eseguita la prima riga scrivo i nuovi insiemi di stati ottenuti (compreso quello vuoto) nella prima colonna (quella degli insiemi degli stati) e proseguo fino a quando non trovo più nuovi stati.
Etichetto tutti gli insiemi degli stati e li scrivo come stati unici, verifico le transizioni che mi portano da un insieme di stati all'altro e costruisco il DFA.

4.5 Pumping Lemma

Il Pumping lemma è un teorema che ci permette di dimostrare che un linguaggio non è regolare.

Capitolo 5

Automi a Pila

Gli automi a pila (PDA - Push Down Automa) sono una settupla definita come segue

$$A = Q, \Sigma, \Gamma, \delta, q_0, Z_0, F$$

Dove:

- Q è un insieme di stati
- Σ è un alfabeto
- Γ è un alfabeto di simboli di stack
- $\delta \in Qx(\Sigma \cup \{\epsilon\})x\Gamma \to P(Qx\Gamma)$
- q_0 è lo stato iniziale
- Z_0 simbolo inizialmente presente sulla pila
- F è l'insieme di stati finali

La definizione è non deterministica. La pila costituisce una memoria di lavoro che utilizzeremo per memorizzare simboli a nostri piacimento (vedremo che sarà fondamentale per poter riconoscere i CF Language).

Osservazione Ci deve essere sempre un simbolo sulla pila, altrimenti l'automa si blocca, l'unico caso in cui la pila è vuota è quando l'automa accetta per pila vuota.

5.1 Accettazione per stato finale o pila vuota

Possiamo costruire due tipi di PDA

- PDA che accettano per pila vuota
- PDA che accettano per stato finale

5.2 PDA Deterministici VS Non Deterministici

Nel caso di **PDA non deterministici** vedremo che entrambe le tipologia hanno la stessa potenza, discorso diverso per i **deterministici**, dato che questi ultimi **non sono in grado di riconoscere tutti i CF Language**, oltretutto posso costruire un automa che accetta per pila vuota solamente nel caso in cui il linguaggio considerato goda della proprietà del prefisso. Quindi i NPDA sono più potenti dei DPDA (è stata spiegata la dimostrazione).

Osservazione Se $L \in Reg$ allora $\exists PDA$ (non deterministico) tale che $L = L(P)$ accettato per stati finali, inoltre $\exists PDAP_n$ non deterministico tale che $L = N(P_n)$ accettato per pila vuota. Infatti un PDA non deterministico è un ϵ-NFA che usa una pila.
Non è detto però che $\exists$ DPDA P_N tale che $L = N(P_n)$.
Per pila vuota i DPDA sono meno potenti dei DFA non riescono ad accettare tutti i linguaggi CF e neanche tutti i Linguaggi Regolari! Per esempio il linguaggio $L_{vv^r} = \{ww^r | w \in 0, 1*\} \in$ CFL, non è accettato da DPDA, mentre viene accettato da NPDA. Questo perchè in questo caso il DPDA non riesce a capire quando iniziare a svuotare la pila, inserendo però un "segnale di mezzo" c $\{wcw^r\}$ che fa capire all'automa quando iniziare a svuotare la pila allora si può creare un DPDA, quando legge c cambia stato e iniziare a svuotare la pila.

5.3 Descrizioni Istantanee - ID

Una ID (o configurazione) è una tripla (q, w, γ) dove:

- $q \in Q$ è lo stato attuale
- $w \in \Sigma^*$ è l'input residuo (ciò che resta da leggere)
- $\gamma \in \Gamma^*$ è il contenuto attuale della pila

Attverso le ID possiamo rappresentare i vari stati del PDA per arrivare a mostrare l'intera esecuzione del PDA (anche i casi dove si blocca).

5.4 Linguaggi accettati dai PDA

I PDA

5.5 Proprietà del prefisso - Prefix-Free Language

Un Linguaggio L ha la proprietà del prefisso (cioè è Prefix Free) se $\nexists x, y \in L$ tali che $x \neq y$ e x è prefisso di y.

Teorema Un linguaggio L è $N(P)$ (accetta per pila vuota) per un DPA se e solo se L gode della proprietà del prefisso (cioè è prefix-free) ed è $L(P')$
Come già riportato a inizio paragrafo, utilizzo questa proprietà per verificare se posso costruire un DPDA che accetta per pila vuota o se sono costretto ad accettare per stato finale.

Esempio Il linguaggio denotato dalla Espressione Regolare (ER) $\{0\}*$ non è prefix-free quindi non posso costruire un DPDA che accetti per pila che lo riconosca.

I DPDA che accettano per pila sono meno potenti di quelli che accettano per stato finale perchè accettano una sottoclasse di linguaggi che vengono accettati per stati finali (quelli appunto Prefix-Free).

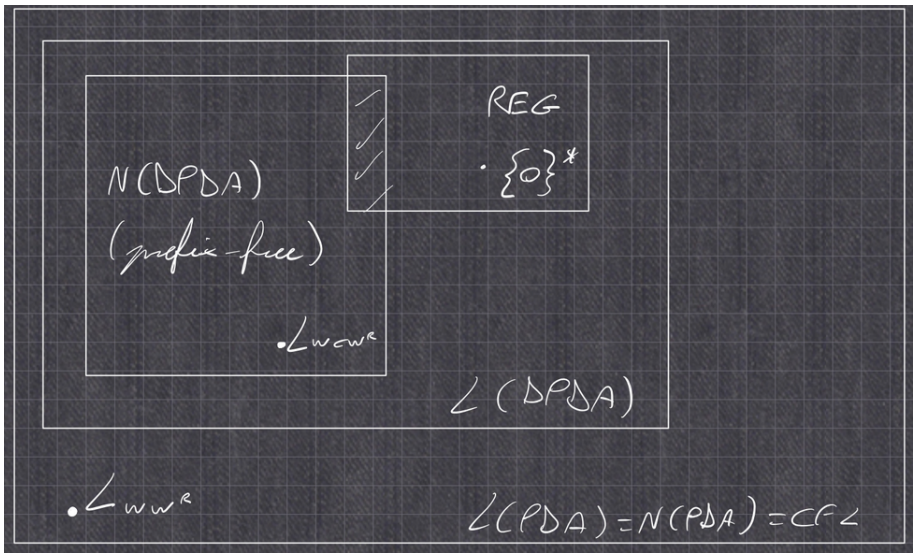
Dimostrazione

Ho un DPDA P che accetta per pila vuota, suppongo che L=N(P) e per assurdo supponiamo che L non sia prefix-free.
Allora $\exists x, y \in L$ t.c $x \neq y$ e x è prefissa di y.
Pongo $y = xw$, se P accetta x dopo aver consumato x, la pila è vuota (dato che x è prefissa di y).
P però non può accettare y anche se $y \in L$ perchè dopo aver consumato x la pila è vuota e dato che il PDA è deterministico e x è prefissa di y l'automa nel leggere y rifarà gli stessi passi di x e quindi arriverà a svuotare la pila prima di aver consumato tutta la stringa.

Riassumendo Per pila vuota i DPDA non sono in gradi accettare tutti i CFL e neanche tutti i LR.

5.6 Riassunto Linguaggi riconosciuti dai PDA

PDA senza D iniziale significa non deterministici.



Capitolo 6

Macchina di Turing

La Macchina di Turing (in seguito abbreviata con MdT), è definita come una macchina astratta costituita da un nastro infinito e una testina che legge ogni carattere del nastro e a seconda della funzione di transizione lo modifica oppure lo lascia invariato e sposta la testina a destra o sinistra.
É definita come una settupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \tag{6.1}$$

- Q - insieme di stati
- Σ - alfabeto di input
- Γ - alfabeto del nastro
- δ - funzione di transizione
- q_0 - stato iniziale
- B - simbolo di blank
- F - insieme di stati finali