Appunti di Automi e Linguaggi Formali

Nicola Baesso

April 17, 2022

Contents

Disclaimer

Questi appunti sono una raccolta parziale delle spiegazioni del prof. Bresolin, e sono state scritte con la gioia di poterle portare al primo parziale dell'anno. Si consiglia comunque di utilizzare solo come eventuale ripasso e non come testo di studio.

1 Introduzione

1.1 Definizioni utili alla comprensione

Per questo corso, ci si avvarrà di alcune definizioni riportate in seguito, utili per una migliore comprensione della materia.

1.1.1 Algoritmi e Problemi

Un problema è definito da 3 (tre) caratteristiche specifiche: l'insieme dei possibili input, l'insieme dei possibili output e la relazione che collega questi due insiemi.
Un algoritmo è una procedura meccanica che, eseguendo delle computazioni eseguibili da un calcolatore, risolve un determinato problema se per ogni input si ferma dopo un numero finito di passaggi e produce un output corretto.
Inoltre è composto da una complessità temporale, che indica il tempo di esecuzione, e una complessità spaziale, che indica la quantità di memoria utilizzata. Entrambe queste misure sono dipendenti dalla dimensione dell'input.

1.1.2 Linguaggi formali

Un linguaggio formale può essere definito come un'astrazione del concetto di problema.
Infatti, un problema può essere espresso sia come un insieme di stringhe (che da qui in avanti indicheremo con Linguaggio), con soluzioni che indicano se una stringa è presente nel Linguaggio o meno, oppure come una trasformazione tra vari Linguaggi, dove la soluzione trasforma una stringa di input in una stringa di output.
Quindi, ogni processo computazionale può essere ridotto ad una determinazione dell'appartenenza ad un insieme di stringhe, oppure essere ridotto ad una mappatura tra insiemi di stringhe.

1.1.3 Automi

Un automa è un dispositivo matematico (inteso in forma astratta) che può determinare l'appartenenza di una stringa ad un Linguaggio e può trasformare una stringa in una seconda stringa. Possiede ogni aspetto di un computer, poichè dato un input provvede a fornire un output, è dotato di memoria e ha la capacità di prendere delle decisioni.
La memoria per un automa è fondamentale. Sostanzialmente esistono automi a memoria finita e automi a memoria infinita, quest'ultimi con accesso limitato e non.
Chiaramente si hanno vari tipi di automi, ognuno di questi adatti ad una determinata classe di linguaggi, dove vengono differenziati per quantità di memoria e per il tipo di accesso ad essa.

1.1.4 Alfabeto e Linguaggio

Esattamente come nel caso del linguaggio naturale, un alfabeto è un insieme finito e non vuoto di simboli, ed è indicato con il simbolo Σ. Da esso si ha la stringa, ovvero una sequenza finita di simboli presi da un alfabeto. Inoltre, si definisce come stringa vuota la stringa senza alcun simbolo preso dall'alfabeto, e si indica con il simbolo ε. Infine la lunghezza di una stringa indica il numero di simboli presenti nella stringa, indicandola con |w|, con w una stringa qualsiasi .

Esempio Si consideri il codice binario, composto da 0 ed 1. Allora definiremo l'alfabeto come Σ={0,1}, una stringa valida come w=010110, e in questo particolare caso |w|=6.

La potenza di un alfabeto, espressa come Σ^k con k>0, esprime l'insieme delle stringhe composte da simboli dell'alfabeto di lunghezza k. L'espressione Σ^* indica l'insieme di tutte le stringhe sull'alfabeto.

Esempio Consideriamo nuovamente il codice binario. Per 0<k<2 (inclusi) si ha
Σ^0=ε (la stringa vuota)
Σ^1={0,1}
Σ^2={00,01,10,11}
E così per ogni k positivo. Mentre $\Sigma^*=\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \dots$

Quindi la potenza di un alfabeto crea a sua volta un alfabeto, con stringhe di lunghezza k, e tale alfabeto è composto da 2^k stringhe.

Un linguaggio L è un sottoinsieme dell'insieme di tutte le stringhe dell'alfabeto. In simboli: L $\subseteq \Sigma^*$ per un certo Σ.

1.1.5 Operazioni sui linguaggi

Definiamo le operazioni che si possono fare sui linguaggi:

-Unione: L $\cup$ M={w : w $\in$ L oppure w $\in$ M}
-Intersezione: L $\cap$ M={w : w $\in$ L e w $\in$ M}
-Concatenazione: L.M={uv : u $\in$ L e v $\in$M}
-Complemento: $\overline{L}$={w : w $\notin$ L}
-Chiusura di Kleene (o Star): L*={w1,w2,...wk : k $\geq$ 0 e ogni wi $\in$ L}

Tutte questa operazioni godono (e ci fanno godere) della proprietà di chiusura. In altre parole, se L e M sono linguaggi regolari, allora anche L $\cup$ M, L $\cap$ M, L.M, $\overline{L}$ e L* sono linguaggi regolari.

Importante! Nelle prossime righe si parleranno di DFA, funzioni di transizioni e quant'altro. Se non si ha la più pallida idea di cosa significhino queste espressioni, v'invito a passare per la sezione DFA del seguente file, e successivamente ritornare sull'esempio che segue.

Esempio Vogliamo (in realtà no, ma ci serve) dimostrare che la proprietà di chiusura vale per l'intersezione (dimostriamo la chiusura per intersezione).
Partiamo dal fatto (che vedremo in dettaglio più avanti) che, essendo L e M linguaggi regolari, esiste un DFA che accetta L e un DFA che accetta M. Quindi, essendo anche L $\cap$ M un linguaggio regolare, possiamo costruire un automa (più precisamente, un DFA) rappresentativo del linguaggio. In particolare, questo automa simula Al che Am in parallelo, e accetta una parola $\Leftrightarrow$ tale parola è accettata sia da Al che da Am.
La funzione di transizione dell'automa è formata da (p,q) $\rightarrow$ (s,t), con p $\rightarrow$ s funzione di transizione di Al, e q $\rightarrow$ t funzione di transizione di Am.
L'automa accetta una parola solo se Al e Am sono entrambi in uno stato finale.
In altre parole, con p stato di Al e q stato di Am, l'automa accetta una parola solo se (p,q) è una coppia di stati finali. Formalmente:

AL$\cap$M=(Ql $\times$ Qm, Σ, δL$\cap$M, (ql,qm), Fl $\times$ Fm)
dove il simbolo $\times$ è il prodotto cartesiano.

L'esempio prende in considerazione il linguaggio L$\cap$M, ma il concetto della costruzione dell'automa è valido anche per dimostrare la proprietà di chiusura nelle altre operazioni.

2 DFA, NFA, espressioni e Linguaggi (non) regolari

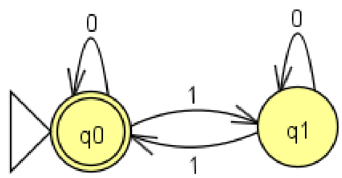
Dopo aver appreso le definizioni necessarie, andiamo a scoprire in questa sezione i primi automi (DFA ed NFA), cosa sono le espressioni regolari e cosa sono i Linguaggi non Regolari.

2.1 DFA

Gli Automi a Stati Finiti (in Inglese Deterministic Finite Automation, abbreviato in DFA) sono la forma più semplice di automa e dispongono di una quantità finita di memoria. Tali automi accettano una parola nel linguaggio unicamente se è in uno stato terminale.

Esempio Trovare ogni parola nel linguaggio binario che sia composta da un numero pari di 1.
Per creare questro DFA, abbiamo bisogno di due stati:
-Il primo sarà lo stato con il numero pari di 1. Tale stato sarà sia iniziale che terminale (e quindi accetterà la stringa).
-Il secondo sarà lo stato nel quale si ha un numero dispari di 1. In tale stato l'automa non accetterà la stringa.

Nel caso s'incontri uno 0, si deve rimanere nello stato attuale.
Segue il diagramma di transizione dell'automa, fatto con JFlap:



Si noti, nel esempio, che ogni stato esegue una transizione **per ogni simbolo nel linguaggio**. Inoltre, quello che fa l'automa è semplicemente "contare" quante cifre di 1 sono presenti, accettando la stringa solo quando questa cifra è pari.
In generale, possiamo dire che un DFA utilizza un numero di stati per "contare" ad un determinato scopo, ed ha una transizione per ogni simbolo del linguaggio che deve analizzare.

In maniera formale, un DFA si definisce con A=(Q,Σ,δ,q0,F), dove Q è un insieme finito di stati, Σ è un alfabeto finito (rappresenta i simboli dati in input al DFA), δ è una funzione di transizione ((q,a)$\mapsto$ q'), q0 è lo stato iniziale (e logicamente è nell'insieme di tutti gli stati dell'automa. In simboli: q0 $\in$ Q), e F è un insieme di stati finali (ovvero gli stati dove l'automa accetta la stringa, in simboli F $\subseteq$ Q).

Come già visto nell'esempio, un DFA può essere graficamente rappresentato con cerchi, ad indicare gli stati, e con frecce, ad indicarne le transizioni. Però un DFA può essere rappresentato anche tramite una tabella, chiamata tabella di transizione.

Esempio Riprendiamo l'esempio precedente.
Abbiamo già espresso l'automa che accetta ogni linguaggio con un numero pari di 1 come diagramma di transizione. Ora rappresentiamolo come tabella di transizione.
Per fare ciò, partiamo dallo stato iniziale, nel nostro caso q0, e ci segniamo le sue transizioni: se si ha uno 0, l'automa resta in q0, mentre se si ha un 1 l'automa va nello stato q1. Per q1 si fa la stessa cosa: se si ha uno 0, l'automa resta in q1, mentre con un 1 l'automa va (in questo caso, ritorna) in q0.
Di seguito, in forma tabellare, quel che ci siamo appena detti:

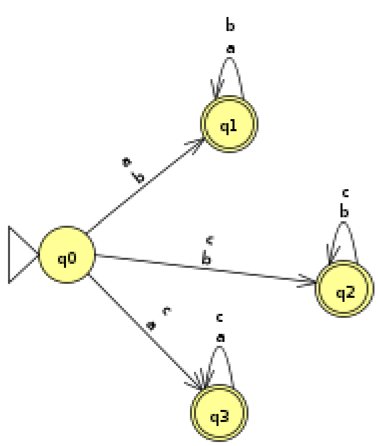
	0	1
$\rightarrow$ q0*	q0*	q1
q1	q1	q0*

Come si nota dall'esempio, nella tabella di transizione si segna con una freccia lo stato iniziale dell'automa, e con un asterisco lo stato finale dell'automa stesso (che nel nostro esempio coincide con lo stato iniziale, ma non è sempre così).
Importante! Un DFA accetta una parola w se la sua computazione accetta w, ovvero l'automa termina in uno stato finale. Inoltre, se il DFA accetta w, allora w appartiene ad un linguaggio regolare.
Più formalmente, si indica con L(A)=$\{$w $\in \Sigma^*$ |A accetta w$\}$ il linguaggio accettato, che è un linguaggio regolare.

2.2 NFA

Gli Automi a Stati Finiti non Deterministici (in inglese Nondeterministic Finite Automation, abbreviato in NFA) sono una forma di automi a stati finiti, che è utilizzato in contesti dove "contare" non è semplice, quindi si lascia spazio al non-determinismo.
Esempio Costruire un NFA che riconosca la parola che, sull'alfabeto $\{$a,b,c$\}$, non sia composto da tutti i simboli.
Per questo esempio, usare un DFA risulterebbe alquanto complesso, quindi ci avvarremo di un NFA per semplicità. Infatti, ci saranno sufficienti 4 stati e 6 transizioni in tutto. Di seguito troviamo sia tabella che diagramma di transizione per il seguente NFA:

	a	b	c
$\rightarrow$ q0	$\{$q1,q3$\}$*	$\{$q1,q2$\}$*	$\{$q2,q3$\}$*
$\{$q1,q2$\}$*	q1*	$\{$q1,q2$\}$*	q2*
$\{$q1,q3$\}$*	$\{$q1,q3$\}$*	q1*	q3*
$\{$q2,q3$\}$*	q3*	q2*	$\{$q2,q3$\}$*
q1*	q1*	q1*	$\emptyset$
q2*	$\emptyset$	q2*	q2*
q3*	q3*	$\emptyset$	q3*

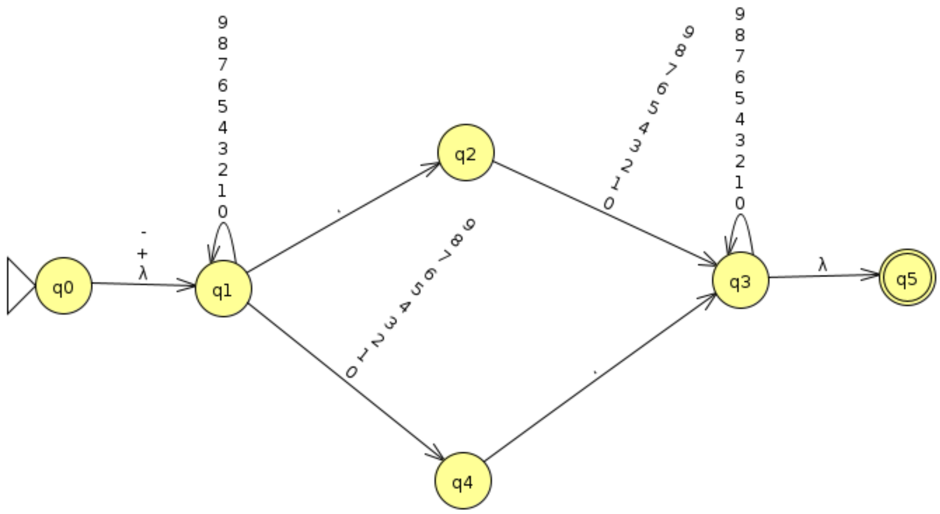


Esattamente come prima, nella tabella rappresentiamo lo stato iniziale con una freccia e con un asterisco rappresentiamo lo stato finale. Generalmente, può capitare che la tabella di transizione può essere "smagrita" ovvero che la tabella completa contiene transizioni a stati mai attraversati dall'automa (non è il caso dell'esempio), e quindi possono essere rimossi. Notiamo però che gli stati inclusi nella tabella non sono semplicemente solo gli stati semplici ma includo anche dei "gruppi" di stati, essendo le transizioni dirette in più stati (grazie al non-determinismo).
Formalizziamo: un NFA NA è descritto con NA=(Q,Σ,δ,q0,F), con l'unica differenza (rispetto ai DFA) che la funzione di transizione restituisce un sottoinsieme di Q.
Importante! Un NFA, a differenza di un DFA, può non arrivare alla fine della computazione ed interrompersi prima (come si nota bene dalle tabelle di transizioni), e quindi accetta una parola w se almeno una sua computazione accetta w, ovvero l'automa termina, in almeno una delle sue computazioni, in uno stato finale. Esattamente come nel DFA, se l'NFA accetta w, allora w appartiene ad un linguaggio regolare (la formalizzazione è la stessa).

2.2.1 ε-NFA

Gli ε-NFA sono NFA che utilizzano le ε-transizioni, ovvero utilizza transizioni che **non consumano il carattere**, quindi l'automa passa allo stato successivo mantenendo la stringa come prima.
Esempio Costruire un ε-NFA che accetti numeri decimali, riconoscendo il segno (opzionale),una stringa di decimali, un punto e un altra stringa di decimali, tenendo conto che una delle stringhe di decimali non può essere vuota.
In questo caso, usare un NFA semplice non è una buona idea, poichè non ci permette di dare opzionalità per quanto riguarda il segno. Inoltre non ci farebbe finire in uno stato finale al termine della lettura dei decimali. Di seguito troviamo sia tabella che diagramma di transizione per il seguente ε-NFA:

	ε	+,-	.	0,..,9
$\rightarrow$ q0	q1	q1	$\emptyset$	$\emptyset$
q1	$\emptyset$	$\emptyset$	q2	{q1,q4}
q2	$\emptyset$	$\emptyset$	$\emptyset$	q3
q3	q5*	$\emptyset$	$\emptyset$	q3
q4	$\emptyset$	$\emptyset$	q3	$\emptyset$
q5*	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Essendo lo stato {q1,q4} non raggiungibile, si omette nella tabella.
Formalmente, un ε-NFA E è definito come E=(Q,Σ,δ,q0,F), dove l'unico parametro differente è la funzione di transizione, che prende in input uno stato in Q e un simbolo nell'alfabeto $\Sigma \cup \{\varepsilon\}$, restituendo un sottoinsieme di Q.
Essendo un particolare tipo di NFA, l'ε-NFA ha tutte le caratteristiche degli NFA.

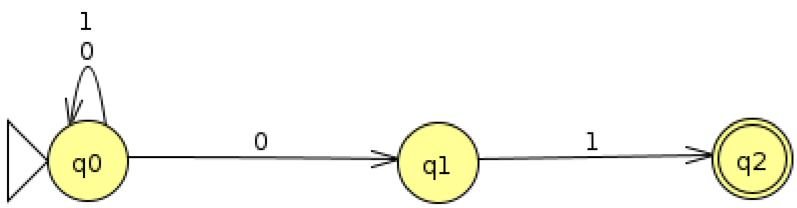
2.3 DFA to NFA (e viceversa)

Sebbene possa sembrare controintuitivo, sia NFA che DFA riconoscono gli stessi linguaggi cambiando unicamente il metodo con cui li riconoscono.
Infatti, $\forall$ NFA N $\exists$ un DFA D : L(D)=L(N) (e viceversa). Per far ciò, si usa un metodo chiamato Costruzione a Sottoinsiemi.

2.3.1 Costruzione a Sottoinsiemi

La Costruzione a Sottoinsiemi ci permette, dato un NFA, di ricavare un DFA che possa riconoscere il linguaggio del NFA. In particolare, ogni stato del DFA corrisponde ad un insieme di stati del NFA, il DFA inizia in {q0}, ovvero nell'insieme di stati con solo q0, nel DFA una stato finale corrisponde ad almeno uno stato finale del NFA, e la funzione di transizione del DFA si riferisce ad uno stato contenuto nel target della funzione di transizione di un NFA.

Esempio Prendiamo il seguente NFA:



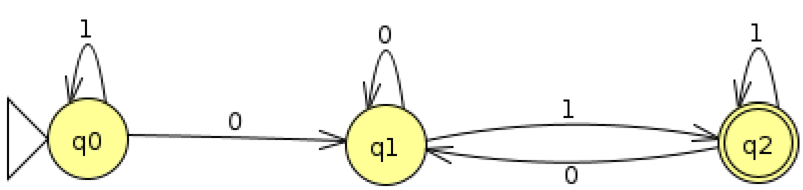
Per "tradurlo" in un DFA, dobbiamo fare una tabella di "conversione", come la seguente:

	0	1
$\rightarrow$ {q0}	{q0,q1}	{q0}
{q1}	$\emptyset$	{q2}*
{q2}*	$\emptyset$	$\emptyset$
{q0,q1}	{q0,q1}	{q0,q1,q2}*
{q0,q2}*	{q0,q1}	{q0,q1}
{q1,q2}*	$\emptyset$	{q2}*
{q0,q1,q2}*	{q0,q1}	{q0,q1,q2}*

Ma se osserviamo, alcuni stati sono inutili, ovvero non contribuiscono in alcun modo alla creazione del DFA, e l'automa non passerà mai per quei stati. Se "puliamo" la tabella otteniamo:

	0	1
$\rightarrow$ {q0}	{q0,q1}	{q0}
{q0,q1}	{q0,q1}	{q0,q1,q2}*
{q0,q1,q2}*	{q0,q1}	{q0,q1,q2}*

Che comprende gli stati che formano realmente il DFA. Se costruiamo il diagramma otteniamo:



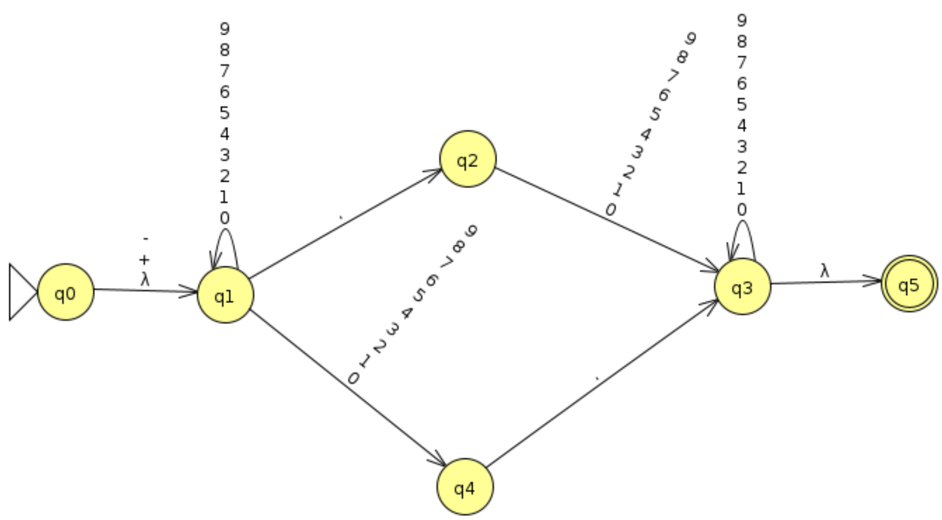
Che è effettivamente corrispondente agli stati e alle transizioni della seconda tabella.
Tale operazione è possibile grazie ad un teorema, che ora enunciamo.

Teorema di equivalenza
Un linguaggio L è accettato da un DFA $\Leftrightarrow$ L è accettato da un NFA

2.3.2 ε-chiusure

Nel caso di ε-NFA, si utilizza una versione modificata della costruzione a sottoinsiemi, caratterizzata dalle ε-chiusure. Con ε-chiusura s'intende l'eliminazione delle ε-transizioni dagli stati che ne fanno uso.
A livello pratico. la costruzione a sottoinsiemi è uguale a quella usata partendo da un NFA, con la variante che lo stato iniziale del DFA è dato dalle ε-chiusure dello stato iniziale del ε-NFA.

Esempio Riprendiamo l'ε-NFA visto in precedenza:



Prima di effettuare la costruzione, dobbiamo effettuare le ε-chiusure:

ECLOSE(q0)={q0,q1}
ECLOSE(q1)={q1}
ECLOSE(q2)={q2}
ECLOSE(q3)={q3,q5}
ECLOSE(q4)={q4}
ECLOSE(q5)={q5}
Dopo questa operazione, possiamo applicare la costruzione. Riportiamo solo la tabella di transizione (per pigrizia):

	+,-	.	0,..,9
$\rightarrow$ {q0,q1}	{q1}	{q2}	{q1,q4}
{q1}	$\emptyset$	{q2}	{q1,q4}
{q2}	$\emptyset$	$\emptyset$	{q3,q5}*
{q1,q4}	$\emptyset$	{q2,q3,q5}	{q1,q4}
{q2,q3,q5}*	$\emptyset$	$\emptyset$	{q2.q3,q5}
{q3,q5}*	$\emptyset$	$\emptyset$	{q3,q5}

Enunciamo anche il teorema per la conversione da ε-NFA a DFA:

Teorema
Un linguaggio L è accettato da un DFA $\Leftrightarrow$ L è accettato da un ε-NFA

2.4 Espressioni Regolari

Abbiamo visto come un linguaggio regolare possa essere espresso tramite un automa a stati finiti. Però non è l'unico modo per descriverlo. Infatti, possiamo esprimerlo anche tramite un'espressione regolare, che non è altro che un modo dichiarativo per descrivere un linguaggio regolare.
Gli ingredienti per costruire un'espressione regolare sono:
-un'insieme di costanti: ε indica la stringa vuota, $\emptyset$ indica il linguaggio vuoto e si usano i simboli presenti in un alfabeto Σ
-operatori: + viene usato per l'unione, . viene usato per la concatenazione e * viene usato per la chiusura di Kleene (* include anche ε)
-parentesi, per il raggruppamento

Con L(E) si indica il linguaggio creato dall'espressione regolare. Essendo una dichiarazione del linguaggio, possono esserci più espressioni corrette per un linguaggio.

Esempio Supponiamo di voler definire come espressione regolare il segeunte linguaggio:

L={w $\in$ {0,1}* : 0 e 1 alternati in w}

Possiamo descriverlo in due modi: o come unione delle possibilità, cioè unione di una alternanza 01, 10, 10101 o 01010, oppure come alternanza di 01, ma con la possibilità di avere un 1 davanti o uno 0.
Poniamolo come espressione, così che risulti più chiaro:

$$(01)^*+(10)^*+1(01)^*+0(10)^*$$

oppure

$$(\varepsilon+1)(01)^*(\varepsilon+0)$$

Si vede come, a livello pratico, non cambi nulla nel significato del linguaggio. Sta quindi a voi scegliere l'espressione che ritenete più opportuna.

3 Grammatiche di Linguaggi liberi da contesto e PDA