Luigi Rizzi (rizzil@unisi.it) - Cristiano Chesi (chesi@media.unisi.it) 4 Marzo 2003

GRAMMATICHE FORMALI E COMPLESSITÀ

Aspetti delle grammatiche generative

(1) I concetti base di lingua, frase ecc. vengono definiti in termini insiemistici:
   a. un **alfabeto** è un insieme finito, non nullo, di simboli
   b. dato un alfabeto A, una **stringa** lunga *n* caratteri può essere definita come una funzione dall'insieme {1,2, ... n} all'insieme A.
      Con **A*** viene solitamente definito l'insieme di tutte le stringhe possibili a partire da A.
      Tra stringhe è definibile come segue l'operazione di **concatenazione**:
      date due stringhe $\varphi = \{(1,x_1),(2,x_2), \ldots (n, x_n)\}$ e $\psi=\{(1,y_1),(2,y_2), \ldots (m, y_m)\}$ l'elemento concatenato sarà definito $\varphi\psi = \{(1,x_1),(2,x_2), \ldots (n, x_n), (n+1,y_1),(n+2,y_2), \ldots (n+m, y_m)\}$
   c. Una **frase** è definita come un insieme ordinato di stringhe.
      ε denota l'elemento nullo (carattere, stringa o frase a seconda dell'insieme di riferimento).
      $\mathbf{A}^+$ è invece l'insieme di tutte le stringhe ad esclusione dell'elemento nullo **ε**.
   d. una **lingua** è un sottoinsieme proprio di A*

(2) Una **grammatica generativa** impiega strumenti matematici, quindi formalmente rigorosi, per descrivere il linguaggio naturale. Ad ogni grammatica (d'ora in poi sarà sottinteso formale) si richiede che sia:
   a. **consistente** cioè nessuna frase può essere contemporaneamente grammaticale e non grammaticale
   b. **esplicita** ovvero i giudizi di grammaticalità devono essere forniti **meccanicamente**, seguendo una derivazione i cui passi sono determinati ed in numero finito, senza far appello ad intuizioni o a conoscenze linguistiche preesistenti ad eccezione della grammatica definita.

(3) **Adeguatezza**: una grammatica deve fornire una descrizione adeguata rispetto alla realtà empirica a cui si riferisce. In particolare si può parlare di adeguatezza a tre livelli:
   a. **osservativa**: la lingua definita dalla grammatica coincide con quella che si intende descrivere
   b. **descrittiva**: l'analisi grammaticale proposta è in linea con le intuizione linguistiche dei parlanti fornendo descrizioni strutturali adeguate delle frasi accettabili
   c. **esplicativa**: i dispositivi generativi utilizzati soddisfano criteri di plausibilità psicolinguistica e riproducono realmente i meccanismi operanti nell'attività linguistica del parlante. Una grammatica si dice esplicativa quando rende conto anche dell'apprendibilità della lingua.

   In genere un numero infinito di grammatiche può essere osservativamente adeguato per una lingua L (ovvero conforme con i giudizi di grammaticalità forniti dai parlanti nativi; l'insieme dei giudizi di grammaticalità fornisce le **assunzioni linguistiche** di base). Data quindi una **classe di grammatiche** adeguata (cioè all'interno della quale sia contenuta almeno una grammatica conforme al modello che generi la lingua L), per escludere grammatiche che non soddisfano il requisito di adeguatezza osservativa si può procedere:
   i. dimostrando che una lingua generata da una classe di grammatica G *deve* avere la proprietà P e che in realtà la lingua L non mostra questa proprietà (modello è in questo caso troppo **generale**)
   ii. dimostrando che una lingua generata da una classe di grammatica G *non può* avere la proprietà P e che in realtà la lingua L mostra questa proprietà (il modello è in questo caso troppo **restrittivo**)
   iii. dimostrandone l'**inadeguatezza condizionale**, ovvero ipotizzando che se esistesse un principio P nella lingua L questo non sarebbe adeguatamente rappresentabile dalla grammatica G (a questo punto l'intuizione andrebbe corroborata da dati empirici effettivamente pertinenti a P per poter determinare l'inadeguatezza di G)

(4) **Decidibilità**: un insieme Σ si dice
   a. **decidibile** o **ricorsivo** se per ogni elemento **e** appartenente all'insieme universo esiste un procedimento meccanico M tale da stabilire in un numero finito di passi se **e** ∈ (appartiene) o ∉ (non appartiene) a Σ (la non appartenenza a Σ determina l'appartenenza al complemento di Σ definito come $\overline{\Sigma}$)
   b. **ricorsivamente enumerabile** quando esiste un procedura che enumera tutti e soli gli elementi di Σ

(5) La **capacità generativa** denota l'insieme di oggetti generati dalla grammatica; tale capacità è:
   a. **debole** se costituita dal semplice insieme di frasi generabili
   b. **forte** se associa a tali frasi l'appropriata descrizione strutturale

(6) Due grammatiche si dicono **equivalenti** se sono in grado di generare lo stesso insieme di oggetti. Si può parlare di **equivalenza debole** o **equivalenza forte**, nella stessa accezione di (5)

(7) Il concetto di **apprendibilità** è intuitivamente associato al fatto che ogni lingua naturale può essere appresa da un bambino esposto ad un sufficiente campione di frasi ben formate. Gold (67) formalizza questo concetto nel seguente modo:

a. un **esempio positivo** x è un'informazione del tipo $x \in L$
b. un **esempio negativo** y è un'informazione del tipo $y \notin L$
c. una **sequenza informativa positiva** è una sequenza di esempi positivi
d. una **sequenza informativa mista** è una sequenza di esempi sia positivi che negativi
e. una **sequenza informativa completa** (**positiva** o **mista**) è una sequenza che enumera tutti gli esempi (solo quelli positivi o esempi misti)
f. un lingua L è **apprendibile** se esiste un algoritmo tale che, per ogni sequenza informativa completa (positiva o mista) esiste un intero *n* tale che, per ogni presentazione di un esempio successivo all'*n*-esimo, l'algoritmo produce in uscita una grammatica di L consistente con tutti gli esempi forniti fino a quel momento

(8) La **grammatica universale** è data dell'insieme dei principi comuni a tutte le lingue. Se l'intuizione chomskyana è corretta non dovrebbero esistere principi specifici per una determinata lingua (ovvero le classiche regole). La variazione interlinguistica dovrebbe poter essere interamente catturata attraverso un numero finito di parametri (auspicabilmente binari) e dalla variabilità lessicale.

Gli universali vengono solitamente distinti in **formali** (dispositivi astratti e generali, quali la necessità di ricorrere a trasformazioni o movimento) o **materiali** (criteri universali di scomposizione, combinazione dei dati empirici, es. "ogni frase può essere rappresentata come l'insieme di una sottosequenza ordinata composta da un sintagma determinante e un sintagma verbale")

(9) Una **grammatica formale G** può essere definita, secondo il modello proposto da Chomsky (65), come una quadrupla ordinata $\mathbf{<V, V_T, \rightarrow, \{S\}>}$ dove:

V è il **vocabolario** della lingua
$\mathbf{V_T}$ è un sottoinsieme di V che racchiude tutti e soli gli **elementi terminali** (il complemento di V_T rispetto a V sarà l'insieme di tutti i vocaboli non terminali e sarà definito come $\mathbf{V_N}$)
$\rightarrow$ è una relazione binaria, asimmetrica e transitiva definita su **V***, detta **relazione di riscrittura**. Ogni coppia ordinata appartenente alla relazione è chiamata **regola di riscrittura**. Per ogni simbolo $A \in V_N$ $\varphi A \psi \rightarrow \varphi \tau \psi$ per qualche $\varphi, \tau, \psi \in V^*$
{S} è un sottoinsieme di V_N definito come l'insieme degli assiomi che convenzionalmente contiene il solo simbolo **S**.

a. Date due stringhe φ e $\psi \in V^*$ si dice che esiste una **φ-derivazione di ψ** se $\varphi \rightarrow^* \psi$.

b. Se esiste una φ-derivazione di ψ allora si può anche dire che **φ domina ψ** ($\varphi \Rightarrow \psi$). Tale relazione è riflessiva e transitiva.

c. Una φ-derivazione di ψ si dice **terminata** se:
   i. $\psi \in V_T^*$
   ii. per nessun χ esiste una ψ-derivazione di χ

d. Data una grammatica G, una **lingua generata** da G, detta **L(G)**, è l'insieme di tutte le stringhe φ per cui esiste una S-derivazione terminata di φ.

(10) una **descrizione strutturale** è un'analisi della stringa che mostra attraverso quali S-derivazioni è stata ottenuta. Le descrizioni strutturali vengono spesso rappresentate da alberi. Un **albero** è un dispositivo formale che può essere descritto come una quintupla **<V, I, D , P, A>** dove:

V è l'insieme finito dei **vertici** (es. v1, v2, v3…)
I è l'insieme finito degli **identificatori** (es. S, DP, VP, la, casa…)
D è la relazione di **dominanza**. È un ordine debole (cioè una relazione binaria, riflessiva, antisimmetrica e transitiva) definita su V
P è la relazione di **precedenza**. È un ordine stretto (cioè una relazione binaria, irriflessiva, antisimmetrica e transitiva) definita su V
A è la **funzione di assegnazione**; una funzione non suriettiva da V a I

Grammatiche, computazioni e complessità

(11) La **forza generativa** delle grammatiche formali, ovvero la capacità che tali sistemi hanno di descrivere (generare) adeguatamente un certo numero di frasi, potrebbe essere intuitivamente intesa come direttamente proporzionale al **numero di frasi** che la grammatica riesce ad accettare. Risulta però immediatamente chiaro, appena in una grammatica si introduce un qualche elemento di ricorsività, che la valutazione della forza generativa deve essere in qualche modo più sottile della semplice numerosità dei fenomeni catturati (che in tutte le grammatiche ricorsive sono infiniti). La forza generativa deve quindi essere calcolata in termini insiemistici: intuitivamente quindi una grammatica G ha una forza generativa maggiore della grammatica G' sse G riesce a catturare le stesse proprietà di G' più almeno una proprietà P che permette a G' di generare espressioni non accettabili in G. Sempre in termini insiemistici si possono definire le **classi di equivalenza** tra grammatiche: la grammatica G e la grammatica G' appartengono alla stessa classe di equivalenza sse sono adeguate (almeno osservativamente) per lo stesso numero di fenomeni.

(12) Per **computazione** si intende la specificazione della **relazione tra un input ed un output**. Questa relazione può essere definita a vari livelli, ma fondamentalmente consiste nella descrizione di una **serie di stadi** intermedi in cui l'informazione di input può venire trasformata prima di raggiungere la forma dell'output e nella definizione delle specifiche di trasformazione. Un problema computazionale tenta quindi di ridurre ogni input ad output seguendo una serie di passi consentiti dal modello computazionale.

(13) La **complessità** di una computazione è direttamente proporzionale alla quantità di risorse impiegate per mappare un input in un output.
Le risorse sono solitamente definite su due dimensioni:
- il **tempo** (**time complexity**), codificato come **passi elementari** da compiere.
- la **memoria** (**space complexity**), come la **quantità di informazione massima** richiesta ad ogni passo.

La complessità è **direttamente proporzionale alla dimensione del problema** (ordinare 1000 parole sarà più complesso che ordinarne 10); di conseguenza si può intuire che la complessità di una grammatica sarà **direttamente proporzionale alla sua forza generativa**.
La **dimensione del problema** (e quindi la sua complessità) viene espressa in funzione dell'**input** da processare quindi l'**ordine di complessità** si potrà esprimere in funzione di tale input. Ad esempio si può dire che un problema ha un ordine di complessità temporale **polinomiale**, se il numero dei passi da compiere per risolverlo, dato un input di dimensione (o lunghezza) n, sarà espresso da una funzione del tipo $c \bullet n^2$ con c che è una costante data (dipendente dal tipo di computazione). Si dirà che tale problema ha ordine di complessità pari a n^2 poiché la costante potrà relativamente influire sulla crescita della funzione per n che tende all'infinito. Tale ordine di complessità si definisce: $O(n^2)$.

(14) Nello studio della complessità, siamo interessati ad individuare il **tasso di crescita** della funzione che esprime il mapping tra input e output in termini di dimensione dell'input. Per problemi "limitati" temporalmente e spazialmente (vedere nell'hand-out successivo il concetto di "spazio del problema") il problema della complessità è relativamente rilevante. Al crescere dell'input a piacimento (n che tendente all'infinito), come nel caso di qualsiasi grammatica formale, il tasso di crescita della funzione è cruciale per determinarne la **trattabilità**: si dice che un problema è trattabile se esiste una procedura che fornisca una soluzione (positiva o negativa) in un tempo finito. Un problema si dice **umanamente trattabile** se l'algoritmo fornisce una risposta in tempi accettabili.
Un problema con ordine di complessità **esponenziale** (es. $O(2^n)$) sarà difficilmente umanamente trattabile. Per avere un'idea intuitiva del tasso di crescita di alcune funzioni, ipotizzando di disporre di una macchina che gestisca un milione di passi al secondo, ecco il calcolo del tempo che occorrerebbe per risolvere tali funzioni:

dimensione input → ↓ funzione	10	20	50	100
N^2	0,0001 secondo	1,0004 sec.	0,0025 sec.	0,1 sec
N^5	0,1 sec.	3,2 sec.	5 minuti e 2 sec.	2 ore e 8 min.
2^N	0,001 sec.	2 sec.	35 anni e 7 mesi	400 trilioni di secoli
N!	3,6 sec.	circa 771 secoli	un numero di secoli a 48 cifre	un numero di secoli a 148 cifre
N^N	2 ore e 8 minuti	più di 3 trilioni di anni	un numero di secoli a 75 cifre	un numero di secoli a 185 cifre

N.B. Il Big Bang è avvenuto approssimativamente 15 miliardi di anni fa…

(15) La tesi di **Turing-Church** ci dice che ogni compito computazionale che può essere realizzato da un qualsiasi dispositivo fisico può essere realizzato anche da una macchina di Turing e che se il dispositivo fisico riesce a completare il compito in F(n) passi, con *n* uguale alla dimensione dell'input, la macchina di Turing ci riuscirà in G(n) passi, con F che differisce da G di al massimo un polinomiale. Questo implica che se una macchina di Turing richiede un tempo esponenziale (cioè t^n) per risolvere un determinato compito, allora non ci sono speranze di trovare un dispositivo fisico qualsiasi che realizzi tale compito.

Calcolare la complessità delle proprietà linguistiche che si intendono catturare è quindi indispensabile per poi scegliere il meccanismo formale che implementerà la grammatica in grado di esprimerle.

(16) Per capire meglio come funziona la complessità di un problema e come la si calcola, prendiamo due classici problemi:

a. il **3SAT problem** è una variante del **problema del soddisfacimento** (**satisfability problem** o **SAT**); il problema consiste nel trovare, se questa esiste, una particolare assegnazione di valori verofunzionali alle lettere proposizionali di una formula booleana in modo che l'intera formula sia vera. La forma che la formula assume è la seguente:
$(\neg a \vee b \vee c) \wedge (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c) \wedge (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (\ldots)$
nel **peggiore dei casi**, si dovrebbero provare tutte le combinazioni di assegnazioni possibili, ovvero $\mathbf{2^N}$ (dove 2 è il numero dei possibili valori dei singoli proposizionali, Vero e Falso, e N è il numero di tali proposizionali, a, b, c…). Il problema risulta quindi temporalmente a crescita esponenziale, ma mostra una caratteristica molto interessante: una volta proposta una soluzione, questa sarà facilissima da verificare! Un problema come 3SAT si dice quindi **difficile da risolvere ma facile da verificare**.
b. il **QBF problem** (problema della Quantificazione di Formule Booleane) è un altro noto problema in cui si chiede di determinare, come in 3SAT, di trovare un'assegnazione di valori per le variabili quantificate in una formula al fine di far risultare l'intera formula vera. La struttura della formula è la seguente:
$Qx_1,Qx_2\ldots Qx_n\ F(x_1,x_2\ldots x_n)$ (con Q uguale a $\forall$ oppure $\exists$.
Il problema è difficile da risolvere, come 3SAT, ma anche difficile da verificare: in effetti, ogni proposizionale di un problema di 3SAT potrebbe essere considerato come quantificato esistenzialmente. Il fatto di introdurre su alcuni proposizionali una quantificazione universale, comporta che ogni assegnazione di valore va verificata, quindi la verifica della soluzione avrà lo stesso ordine di complessità del problema 3SAT.

(17) Ipotizzando che un computer riesca a risolvere effettivamente un problema come il 3SAT, tale computer dovrà utilizzare un algoritmo al peggio polinomiale. Ma vista la natura del problema, tale algoritmo dovrà necessariamente essere non-deterministico. Tale algoritmo potrà essere implementato in quella che viene chiamata una **Macchina di Turing** (**MT**) **non-deterministica**.

Questi problemi vengono definiti di tipo **NP**, che significa **Non-deterministic Polynomial time**.

I problemi di ordine **P** sono invece deterministici e polinomiali in funzione del tempo. La classe dei problemi in P è inclusa in quella dei problemi in NP, ma per quanto riguarda i problemi come 3SAT in NP, non esiste prova concreta della loro **riducibilità** (vedi (18)) a problemi di ordine P.

Questo tipo di problemi si dice **NP-hard**, poiché presentano le stesse difficoltà dei problemi della classe NP. Se, come nel caso di 3SAT, si può ipotizzare che un ipotetico dispositivo "preveggente" possa risolvere il problema con una funzione di ordine polinomiale si dice che il problema è **NP-completo**. Un chiaro segnale di NP-completezza è il fatto che tale problema sia difficile da risolvere ma facile da verificare una volta proposta una soluzione.

Un'altra classe di problemi è quella che richiede uno spazio (o memoria) polinomiale su una MT (tale classe viene definita **PSPACE**). È interessante notare che la classe dei problemi NP è inclusa nella classe PSPACE (la dimostrazione di questa relazione insiemistica risiede nel fatto che lo spazio, contrariamente al tempo può essere riusato durante la computazione). Il QBF è un problema che appartiene a questa classe.

(18) Per scoprire la classe di complessità di un problema si ricorre alla **riduzione**: si prende cioè un problema di cui si conosce già la complessità, si trova un mapping efficiente che trasformi ogni istanza del problema noto in un istanza del nuovo problema e che preservi i risultati richiesti.
È questo il caso dell'**Universal Recognition Problem** (**URP**) che viene così formulato:
Data una grammatica G (in qualsiasi framework grammaticale) e una stringa α, α appartiene al linguaggio generabile da G?

L'URP è un problema di parsing generalizzato che può essere ridotto ad un problema come 3SAT. Senza addentrarci nella dimostrazione, l'intuizione di base consiste nel fatto che per una stringa α, come un proposizionale a in una formula di 3SAT, può esistere un'assegnazione ambigua di valori (ad esempio la stringa *vecchia* può essere sia nome che aggettivo, mentre il proposizionale a può essere vero o falso). Inoltre si può parlare di qualcosa come la verifica di accordo sia per la stringa α (banalmente accordo in senso linguistico) che accordo tra i proposizionali in una formula 3SAT (inteso come consistenza dell'assegnazione di valori). Si deduce quindi che l'URP è almeno complesso quanto il 3SAT e che quindi è **NP-completo**.

(19) La **gerarchia di Chomsky** è un utile strumento per definire i rapporti insiemistici tra lingue generate da grammatiche di complessità differente.
Tali relazioni mostrano che:

a. una grammatica di potere generativo maggiore sussume quelle di potere generativo minore (il rapporto di inclusione esprime questo concetto)
b. la riduzione di potere generativo di una grammatica viene ottenuta ponendo maggiori vincoli alle regole di riscrittura

Gli schemi di regole che Chomsky utilizza per definire la gerarchia sono i seguenti:

tipo 0	(**Touring equivalenti**)	$\boldsymbol{\alpha \rightarrow \beta}$ **(con** $\boldsymbol{\alpha \neq \varepsilon}$**)**
tipo 1	(Dipendenti dal Contesto, **Context Sensitive**)	$\boldsymbol{\alpha A \beta \rightarrow \alpha \gamma \beta}$ **(con** $\boldsymbol{\gamma \neq \varepsilon}$**)**
tipo 2	(Indipendenti dal Contesto, **Context Free**)	$\boldsymbol{A \rightarrow \gamma}$
tipo 3	(**Regolari**)	$\boldsymbol{A \rightarrow xB}$ oppure $\boldsymbol{A \rightarrow x}$

dove:
$\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$ possono essere sequenze di simboli terminali e non terminali (compreso l'elemento nullo ε salvo diversa specificazione)
A, **B** sono singoli simboli non terminali
x è una sequenza di simboli terminali

- i **linguaggi regolari** (o **linguaggi di tipo 3**), rappresentano le grammatiche più restrittive e possono essere **lineari a destra** oppure **lineari a sinistra.**
  In entrambi i casi, la parte sinistra della regola deve contenere un singolo simbolo non terminale e la parte sinistra al più un simbolo non terminale. Nel caso di lingue **lineari a destra** se la parte destra della regola ha un simbolo non terminale esso sarà necessariamente l'ultimo simbolo della stringa ($B \rightarrow abA$); viceversa se il linguaggio è **lineare a sinistra** il simbolo non terminale, se presente nella parte destra della regola, sarà il primo alla sinistra della stringa ($B \rightarrow Aab$).
- i **linguaggi indipendenti dal contesto** (**Context-Free** o **linguaggi di tipo 2**) permettono di riscrivere ogni singolo simbolo non terminale con una sequenza qualsiasi di simboli terminali e non terminali oppure con ε.
- i **linguaggi dipendenti dal contesto** (**Context-Sensitive** o **linguaggi di tipo 3**) seguono la regola implicita di "non riscrivere mai sequenze di simboli utilizzando un numero inferiore di caratteri di quello utilizzato alla sinistra della regola". Concretamente, dato un contesto delimitato dalle sequenze arbitrarie di simboli $\boldsymbol{\alpha}$ e $\boldsymbol{\beta}$, ogni singolo simbolo non terminale può essere riscritto come una sequenza non nulla di simboli terminali e non terminali arbitrari. Le grammatiche le cui regole non mostrano **ricorsività** (un simbolo non terminale nella parte sinistra della regola viene riscritto nella parte destra, ad esempio $\alpha A \beta \rightarrow \alpha \alpha A \beta$), risultano equivalenti ai linguaggi regolari.
- i **linguaggi senza restrizioni** (o **linguaggi di tipo 0**) sono generati dalle grammatiche con la maggiore forza generativa. Nessun vincolo viene posto ad eccezione del fatto che la parte sinistra della regola non debba essere nulla (ε). Tali linguaggi godono della proprietà di essere **ricorsivamente enumerabili** (e quindi **Turing computabili**).

La gerarchia di Chomsky può essere quindi rappresentata insiemisticamente nel seguente modo:

Linguaggi di tipo 3 - Espressioni Regolari e Automi a Stati Finiti (Finite State Automata - FSA)

(20) Le **espressioni regolari** sono un primo semplice esempio di linguaggio di tipo 0 e rappresentano una notazione algebrica per definire insiemi di stringhe di testo.
Il cuore dell'espressione regolare è il **pattern di identificazione** composto da caratteri alfanumerici (compresi segni di spaziatura e di interpunzione) e da segni speciali volti a stabilire le relazioni tra i caratteri del pattern.

Ecco come convenzionalmente vengono usati i caratteri speciali:

Espressione Regolare	Corrispondenza	Esempio di pattern identificato
[Dd]uomo	Duomo oppure duomo	Il duomo è nella piazza
[a-z]	qualsiasi carattere alfabetico minuscolo	Il duomo è nella piazza
[A-Z]	qualsiasi carattere alfabetico maiuscolo	Il duomo è nella piazza
[0-1]	qualsiasi carattere numerico	Il numero 1 di vicolo dell'Oro
[^a-z]	tutto fuorché lettere minuscole	Il numero 1 di vicolo dell'Oro
sali?ta	salita oppure salta	Marco deve saltare
sal.ta	accetta qualsiasi carattere tra le *i* e la *t*	Marco saluta
bu*	b seguito da un numero imprecisato (anche nullo) di *u*	buuuuuu! oppure b!
bu+	b seguito da un numero imprecisato (ma non nullo) di *u*	buuu!
bu{n,m}	b seguito da un numero di u compreso tra n ed m (se m non viene specificato il numero di *u* è uguale a n)	l'espressione "bu{1,3}" accetta bu!, buu! e buuu!
^L	il diacritico ^ segna l'inizio della stringa; l'espressione restituisce quindi le sole *L* che iniziano una stringa	La casa
a$	il diacritico $ segna la fine della stringa; l'espressione restituisce quindi solo le *a* che terminano una stringa	La casa
cas(a|e)	il pipe | è equivalente alla disgiunzione logica le parentesi restringono la portata dell'operatore, in questo caso la stringa *cas* può essere seguita o da *a* o da *e*	Marco vive in un casale
*	il backslash è il simbolo di escape, che permette cioè di cercare come testo semplice i caratteri speciali (in questo caso l'asterisco)	A*

(21) Le espressioni regolari sono principalmente usate per cercare file, parole in un testo, documenti su web. Ma il loro successo nell'ambito del NLP è storicamente dovuto all'operazione di **sostituzione** applicata a sistemi conversazionali quali **Eliza** (Weizenbaum 66).
La sostituzione è un'operazione che permette di sostituire l'occorrenza di un'espressione regolare con un'altra espressione regolare e può essere definita come segue:
s/espressione_regolare1/espressione_regolare2/ (es. s/ www\.[a-z]\.com / www\.pe{2}\.com/ permette di sostituire il dominio di un qualsiasi sito web con la parola *pepe*. www.sale.com > www.pepe.com).
Un'utile estensione delle espressioni regolari è l'uso dei **registri**: se si usano più blocchi di operatori (ogni parentesi tonda delimita un blocco), si può riutilizzare l'espressione trovata da un determinato blocco nell'espressione da sostituire, facendo riferimento all'ordine dei blocchi nel pattern di ricerca:

es. s/ la (casa|macchina) è stata comprata da (Maria|Gianni)/ \2 ha comprato la \1 /
permette di costruire la forma attiva (Gianni ha comprato la casa) della frase passiva (la casa è stata comprata da Gianni).

Eliza simulava uno psicologo rogeriano, in grado di condurre una conversazione sullo stile di quella che segue:

Utente: gli uomini sono tutti uguali
Eliza: in che senso?
Utente: ci seccano sempre con una scusa o l'altra
Eliza: potresti far riferimento ad un esempio specifico?

Utene: beh, il mio ragazzo mi ha fatto venire qui.
Eliza: il tuo ragazzo ti ha fatto venire qui
Utente: dice che sono sempre depressa
Eliza: sono spiacente di sapere che sei depressa

Eliza applica una cascata di espressioni regolari all'input dell'utente e lo trasforma, tramite sostituzione, nella sua domanda. Queste alcune delle regole utilizzate da Eliza:

s/_sono_[.*_ |](depress[o|a]|triste)/sono spiacente di sapere che sei \1/
s/_sono_tutt[i|e]_(.*)_/in che senso sono \1?/
s/_sempre_/ potresti far riferimento ad un esempio specifico?

(22) Le espressioni regolari (escludendo l'estensione dei registri) permettono di descrivere grammatiche regolari e risultano quindi implementabili sotto forma di **automi a stati finiti**, **FSA**. Esistono compilatori (famoso è quello sviluppato dalla Xerox) che effettuano una traduzione meccanica tra i due formalismi.

(23) **Determinismo e non determinismo**: come visto in alcuni casi di analisi morfologica l'automa può trovarsi in uno stato dal quale, con lo stesso simbolo in input, può contemporaneamente raggiungere due o più stati distinti (prima fonte di non-determinismo), oppure può essere presente un arco che connette due stati attraverso una transizione ε (seconda fonte di non-determinismo). Un automa **non-deterministico** è chiamato **NFSA** (**Non-deterministic Finite-Stata Automata**) mentre uno **deterministico** si chiama **DFSA**. Intuitivamente il non-determinismo sembra rendere i NFSA degli strumenti generativamente più potenti dei DFSA; in realtà questo non è corretto: per ogni macchina NFSA esiste un algoritmo di conversione che lo trasforma in un DFSA (ed al peggio il numero di stati del DFSA è di 2^N dove N è il numero degli stati del NFSA).

(24) definita la **classe di equivalenza** tra FSA (NFSA o DFSA), espressioni regolari e **linguaggi** (o **insiemi**) **regolari** è appropriato concentrarsi adesso sulle proprietà di questi ultimi, definendoli formalmente come segue:

i. ø (l'insieme vuoto) è un linguaggio regolare
ii. $\forall a \in \Sigma \cup \varepsilon$, $\{a\}$ è un linguaggio regolare
iii. se L_1 e L_2 sono linguaggi regolari allora lo sono anche:
   a. $L_1 \circ L_2 = \{xy |\ x \in L_1, y \in L_2\}$ la loro **concatenazione**
   b. $L_1 \cup L_2$ la loro **unione** o **disgiunzione**
   c. L_1^* oppure L_2^* la **chiusura di Kleene** su L_1 o L_2

Per dimostrare l'equivalenza tra linguaggi regolari e FSA un passo fondamentale è dimostrare che per ogni operazione base esiste un meccanismo a stati finiti che la rappresenta. Ad esempio si possono descrivere i seguenti dispositivi a stati finiti che implementano le tre proprietà fondamentali dei linguaggi regolari appena descritte:
concatenazione – dati due FSA, FSA_1 e FSA_2 la loro concatenazione è data associando ogni stato finale di FSA_1 allo stato iniziale di FSA_2 attraverso transizioni ε.
chiusura – dato un FSA, ogni suo stato finale deve essere connesso con una transizione ε allo stato iniziale
unione – dati n FSA, si crea un nuovo stato esterno q_u dal quale si fanno partire tutte le transizioni ε necessarie per connettere q_u a tutti gli stati iniziali degli n FSA.

(25) Un **linguaggio naturale** è equivalente ad un **linguaggio regolare**, ovvero la sua grammatica può essere descritta attraverso espressioni regolari o automi a stati finiti?

Intuitivamente è facile vedere i problemi che questa equivalenza comporterebbe (vedi Eliza). Inoltre si dovrebbe scomporre la domanda in modo più euristico: esiste una parte delle lingue naturali, ad esempio la fonologia o parte della morfologia, che si comportano come linguaggi naturali?
Per mostrare che un linguaggio è regolare si dovrebbe costruire un'espressione regolare che lo rappresenti e per sapere se questo è teoricamente possibile (visto che intuitivamente sembra piuttosto complesso) si può ad esempio ricorrere alla verifica della proprietà di chiusura del linguaggi regolari rispetto alle operazioni di unione, concatenazione, chiusura di Kleene ecc.

Oppure si potrebbe ragionare inversamente (inadeguatezza condizionale) provando che una proprietà che il linguaggio dovrebbe possedere, ispirata a genuini dati linguistici, in realtà non è catturabile dai linguaggi regolari. Scegliamo questa seconda strada per dimostrare che le lingue naturali non sono linguaggi regolari.

Prima però introduciamo un utile strumento di analisi che ci permetterà di dire quando un linguaggio è regolare:

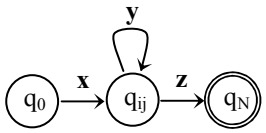
(26) Il lemma del gonfiaggio (**Pumping Lemma**) afferma, informalmente, che se un linguaggio infinito è regolare, allora esiste una qualche stringa che può essere "gonfiata" appropriatamente e indefinitamente (es. nel linguaggio *ab*a* la *b* può essere "gonfiata" a piacimento: *abbbbbbbbba*). Il lemma serve ad identificare i linguaggi infiniti non regolari per i quali questo "gonfiaggio appropriato" non è possibile.

N.B. dimostrare che per un linguaggio vale il Pumping Lemma e cioè che una qualche stringa può essere gonfiata appropriatamente non significa aver dimostrato che tale linguaggio è regolare; ma al momento questo aspetto ci interessa relativamente

La formulazione del Pumping Lemma parte da due considerazioni:

i. se un linguaggio infinito è modellabile con un FSA allora tale linguaggio sarà decidibile, cioè si potrà dire utilizzando una serie di risorse di memoria finite, se una stringa è accettabile oppure no; questo però comporta che, visto che le stringhe possono essere di lunghezza arbitraria, la memoria richiesta non può crescere proporzionalmente con la lunghezza della stringa (intuitivamente si può già prevedere che linguaggi del tipo a^nb^n possano non essere regolari)
ii. il fatto che le stringhe possano essere di lunghezza arbitraria e che gli stati siano necessariamente in numero finito, predice l'esistenza di un qualche loop senza il quale il linguaggio non potrebbe essere regolare.

Prendiamo, ad esempio, un linguaggio L generabile da un FSA a N stati. Una stringa lunga N+1, richiederà necessariamente che almeno due stati (chiamiamoli q_i e q_j) combacino con lo stesso stato. In altri termini su di uno stato q_{ij} deve esistere un loop che permetta all'automa di non consumare un numero di stati maggiore di N. Assumiamo che *x* sia la serie di simboli che la macchina riceve in input da q_0 fino a q_{ij} e *y* la stringa di loop su q_{ij} e *z* l'input per uscire dal loop e passare da q_{ij} a q_N. L'FSA che rappresenta questo concetto è il seguente:



L'espressione regolare equivalente è la seguente: xy*z (che accetta cioè le stringhe xz, xyz, xy^nz)

Usando il Pumping Lemma si può dimostrare che l'intuizione riguardo al linguaggio definito dall'espressione a^nb^n era giusta, ovvero che tale linguaggio non è regolare. La dimostrazione si esegue con tre passi:

a. ipotizziamo per assurdo che il loop y sia composto da sole *a* (quindi per la simmetria dell'espressione x deve essere uguale ad *a* e z uguale a *b*). Questo implicherebbe che l'espressione xy^nz avrà un numero maggiore di *a* rispetto a xyz. Ma visto che il loop è l'unico modo per "pompare" caratteri nell'espressione, il numero di *b* sarà uguale in xy^nz ed in xyz. Dunque in xy^nz ci sono più *a* che *b*;
b. con lo stesso ragionamento si può dimostrare che anche l'ipotesi di loop composto da sole *b* è inconsistente;
c. non resta che pensare che y sia composto sia da *a* che da *b*. Ma questo implicherebbe che la xy^nz presenti *b* prima delle *a* e questo è di nuovo inconsistente con il linguaggio a^nb^n.

Questo strumento ci permette quindi di investigare l'inadeguatezza condizionale delle espressioni regolari per rappresentare il linguaggio naturale come segue nei prossimi esempi:

(27) Un linguaggio regolare non cattura **espressioni speculari**; il linguaggio naturale mostra proprietà speculari.
Chomsky (56, 57) osserva che molte lingue naturali mostrano costruzioni coordinate di questo tipo:
se … allora …
né … né …
queste strutture possono combinarsi teoricamente in modo illimitato seguendo però precisi vincoli strutturali:

se non ci sarà *né* vento *né* nuvole *allora* andrò al mare
**se* non ci sarà *né* vento *allora* andrò al mare *né* nuvole

In particolare sembra impossibile stabilire dipendenze incrociate tra gli operatori.
Queste costruzioni, che rappresentano un sottoinsieme proprio di molte lingue naturali, sono isomorfe ai linguaggi speculari così definiti:

xx^R , x∈ ab*
dove x^R è l'immagine speculare di x.
Per dimostrare che questo linguaggio non è regolare ricorriamo nuovamente al Pumping Lemma. Il linguaggio xx^R può essere descritto da una stringa costruita nel seguente modo:
$x_1,x_2 \ldots x_{n-1},x_n^2,x_{n-1} \ldots x_2,x_1$
un caso speciale di questo linguaggio è quello che riduce la sottostringa $x_1,x_2 \ldots x_{n-1}$ ad una serie di *a* di lunghezza arbitraria (cioè a^n con $n \geq 1$). Il caso speciale avrà quindi la forma $a^nb^2a^n$. La dimostrazione dell'inadeguatezza delle espressioni regolari a rappresentare questo linguaggio che è un sottoinsieme proprio del linguaggio xx^R segue trivialmente i passi della dimostrazione in (26).

Partee estende l'osservazione a strutture ad **incassamento centrale**: un linguaggio regolare è inadeguato per descrivere l'incassamento centrale; il linguaggio naturale mostra proprietà di incassamento centrale:

al gatto che il cane cacciò piace il tonno

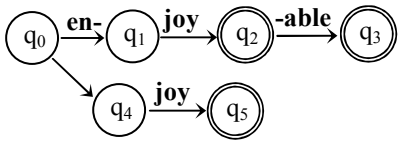
È interessante notare che in realtà anche se la teoria linguistica prevede infiniti incassamenti possibili, in realtà al terzo incassamento la leggibilità della frase degrada drasticamente:

?al gatto che il cane che il padrone comprò, cacciò, piace il tonno.

(28) Un'ulteriore prova (questa volta non formale) di inadeguatezza dei linguaggi di tipo 3 per rappresentare le lingue naturali può essere fornita sulla base della loro **ineleganza** e **implusibilità psicolinguistica**.

Pullum e Gazdar (82) mostrano ad esempio come l'**accordo a lunga distanza** (qual[i|e] problem[a|i] dice il tuo professore [è|sono] irrisolvibil[e|i]?) possa essere teoricamente gestito da FSA, ma solo a scapito di un'enorme esplosione del numero di regole.

Un argomento simile è fornito da Sproat (93) stavolta riguardo all'adeguatezza dei linguaggi regolari nel rappresentare certe proprietà morfologiche. Ad esempio in inglese certi suffissi, quali -able, possono talvolta attaccarsi al verbo solo se questo viene prima prefissato con en- (es. en-joy-able, *joy-able). Questo fenomeno può essere gestito da FSA, ma solo ricorrendo a due rappresentazioni distinte del morfema joy:



(29) **proprietà utili**: gli FSA permettono di gestire adeguatamente la relazione di **precedenza lineare**.

Linguaggi di tipo 2 – Context-Free Grammars o grammatiche a struttura sintagmatica

(30) Il principale limite dei linguaggi di tipo 3 è spesso identificato con la forte difficoltà nel render conto di un fenomeno strutturale fondamentale nel linguaggio naturale: la strutturazione in **costituenti**.
Un costituente è un'entità strutturale fondamentale composta da più parole che si comporta come un'unità rispetto a determinate regole grammaticali.
Ad esempio l'espressione "il primo maggio" può ragionevolmente essere considerata un costituente sintattico (Sintagma Nominale o NP):
il primo maggio è la festa dei lavoratori
la festa dei lavoratori è il primo maggio
*il primo è la festa dei lavoratori maggio
*il è la festa dei primo lavoratori maggio

(31) I sistemi formali più usati per catturare questa intuizione sono le **Context-Free Grammars** (**CFG**, talvolta chiamate anche Grammatiche a struttura sintagmatica, **Phrase-Structure Grammars** acronimizzato in **PSG**).
Una CFG consiste in un **lessico** di parole e simboli ed un insieme di **regole di produzione** ognuna delle quali esplicita come i simboli del linguaggio possano essere raggruppati e ordinati insieme.
Formalmente una CFG può essere espressa da una quadrupla **<N, Σ, P, S>** tale che:

N è un insieme finito di simboli non terminali (o variabili)
Σ è un insieme finito di simboli terminali (con Σ disgiunto da N)

P è un insieme finito di regole di produzione del tipo A→α, dove A∈ N ed α è una stringa finita composta da simboli∈ (N ∪ Σ)
S è il simbolo iniziale (Sentence)

Una **derivazione** è un'applicazione di una o più regole di riscrittura.
Si dice che **A deriva α_n** se esistono una serie di regole di produzione tali che **$A \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$** .
Se esiste una regola del tipo **$A \rightarrow \alpha_n$** allora si dice che **A deriva direttamente α_n** .
Una lingua L_G, generata da una CFG **G**, è l'insieme delle stringhe composte da simboli terminali che possono essere derivate da S. Ovvero:

L_G = W|w è in Σ* e S →* w

(32) Si chiama **Chomsky Normal Form** (**CNF**, Chomsky 63) una particolare CFG, in cui:
i. $\varepsilon \notin (N \cup \Sigma)$
ii. tutte le regole sono nella forma A → B C oppure A → a (dove A, B, C∈ N e a∈ Σ)

In pratica questo garantisce la binarietà della ramificazione (eccetto per la riscrittura dei nodi terminali).
Inoltre è possibile dimostrare che ogni CFG ha una CNF debolmente equivalente, nel senso di (6): ad esempio ogni regola del tipo A → B C D può essere riscritta con due regole del tipo A → B X , X → C D).

(33) Ecco un esempio di grammatica:
N ={D,N,V,DP,VP}
Σ = {il,cane,gatto,morde,scaccia}
P = {D→il , N→cane, N→gatto,V→morde, V→scaccia, DP→D N,VP→V DP, S→DP VP}
S = {S}

questa grammatica permette di generare una linguaggio in cui le seguenti espressioni risultano ben formate:
il cane morde il gatto
il cane scaccia il gatto
il gatto morde il cane
il gatto scaccia il cane

e di escludere espressioni come:
*il morde cane (poiché non esiste una regola che permette di riscrivere un DP come D oppure come N)

(34) trattamento di fenomeni grammaticali particolari:
a. **accordo**
se nel lessico Σ della grammatica proposta in (33) aggiungiamo le parole {i,cani,gatti, mordono, scacciano} e le regole {D→i , N→cani, N→gatti, V→mordono, V→scacciano} frasi come "i gatto mordono il cani" sarebbero perfettamente accettabili contrariamente a quanto suggerito da chiare intuizioni linguistiche. Per gestire l'accordo, l'unico dispositivo di cui le CFG dispongono è l'introduzione di nuovi simboli non terminali che colgano una più sottile distinzione categoriale. In questo senso si devono introdurre regole più specifiche del tipo {Dpl→i , Npl→cani, Dsg→il, Nsg→cane, DP→ (Dsg Nsg | Dpl Npl)}
Lo stesso problema si presenta per l'accordo soggetto-verbo e verbo-complemento.
b. **sottocategorizzazione**
ci si riferisce allo schema di sottocategorizzazione come alla possibilità di distinguere ulteriormente, all'interno di categorie maggiori (ad esempio la categoria verbale), categorie più precise che rendano conto in modo più adeguato del comportamento dei diversi elementi lessicali (ovvero i simboli terminali in questo framework). Ad esempio mentre "fare" richiede un complemento diretto, "cadere" no. Transitivo, intransitivo, inaccusativo o ergativo sono solo tre delle sotto-classi verbali che servono a predire un determinato comportamento verbale (certi approcci, Levin 83, distinguono fino a 183 classi verbali distinte).
Al di là dei dettagli tecnici, gli schemi di sottocategorizzazione hanno indubbi vantaggi e dovrebbero poter essere implementabili in una grammatica che pretende di essere adeguata per le lingue naturali.
Come per l'accordo, le CFG richiedono la specificazione di sottocategorie ad hoc, come riscritture possibili partendo dalle categorie principali (es. { VP → ($V_{transitivo}$ DP | $V_{intransitivo}$ | $V_{frasale}$ S | V_{modale} V_{inf}…)}
c. **ricorsività**
nulla vieta di poter scrivere regole per le CFG in cui a destra e a sinistra si trovi lo stesso simbolo non terminale (es. S → VP S). Questa proprietà delle CFG è cruciale per distinguerle dagli FSA (Chomsky

dimostra che una CFG le cui regole non contengono nessuna ricorsività risulta debolmente equivalente ad un FSA).
Utilizzando questa possibilità, siamo adesso in grado di gestire i linguaggi del tipo a^nb^n attraverso le due semplici regole che seguono:
$S \rightarrow a\ S\ b$
$S \rightarrow \varepsilon$

Allo stesso modo si riesce a risolvere il problema dell'incassamento centrale posto nella forma $a^nb^2a^n$:
$S \rightarrow a\ S\ a$
$S \rightarrow bb$

e nella sua forma più generale (se… allora…)
$S \rightarrow a\ S\ b\ S$

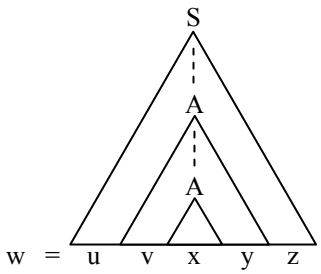
(35) Un **linguaggio naturale** è equivalente ad un **linguaggio context-free**, ovvero la sua grammatica può essere descritta attraverso CFG?

Le uniche prove non ancora confutate che attestano la non equivalenza sembra siano quelle per le **cross-serial dependencies**, fenomeno produttivo nel dialetto svizzero-tedesco che ne attesta l'inadeguatezza per la sintassi, e il Bambara che attesta l'inadeguatezza del modello per la morfologia. L'argomento fondamentale è collegato al fatto che esistono in queste lingue stringhe identiche concatenate tra loro come segue:
$x_1, x_2 \ldots x_n \ldots y_1, y_2 \ldots y_n$

Nel dialetto parlato a Zurigo, è produttivo inserire un argomento al dativo, seguito da un argomento accusativo, seguito da un verbo che richiede un complemento dativo, seguito da un verbo che richiede un complemento accusativo (Shieber 85 mostra come questo comportamento sia produttivo anche con tre dipendenze cross-seriali):
Jan säit das mer em Hans es huus hälfed aastriiche
J. dice che noi a H. la casa abbiamo aiutato a dipingere

(36) Per dimostrare formalmente questa intuizione si esiste un **Pumping Theorem** anche per le **CFG**. Ecco alcune considerazioni utili per introdurlo:

a. ad ogni derivazione che utilizza una CFG può essere associato un **albero** come descritto in (10);
b. l'**ampiezza** di tale albero sarà **limitata dalla sua profondità** (o **altezza**, se si preferisce): in effetti in ogni grammatica le regole potranno riscrivere un simbolo non terminale necessariamente con un numero finito di simboli. Ipotizziamo che la regola più lunga riesca a riscrivere un non terminale con 5 simboli (questo equivale a dire che la parte destra della regola sarà lunga 5 caratteri); quando questa regola viene applicata, l'albero cresce di uno in profondità (o altezza) e al massimo di cinque ramificazioni in ampiezza. Reiterando l'applicazione della stessa regola per ogni nodo avremo al massimo 25 ramificazioni all'espansione successiva; quindi possiamo generalizzare che nel caso più sfortunato l'ampiezza dell'albero sarà al più uguale a n^h con h uguale all'altezza dell'albero e n uguale al numero massimo degli elementi nella parte destra della regola.
c. il numero di simboli non terminali all'interno di una qualsiasi CFG dovrà essere necessariamente finito per definizione, ipotizziamo che sia m. Ma a questo punto se un albero raggiunge ampiezza maggiore di n^m avremmo la garanzia che durante tale derivazione qualche simbolo non terminale dovrà essere ripetuto almeno due volte (chiamiamolo A).
d. sappiamo che ogni derivazione termina con una stringa di simboli terminali (chiamiamola w) e che tale stringa dovrà essere dominata da S. Allo stesso modo ogni nodo non terminale all'interno dell'albero dovrà dominare una porzione di w. In particolare siamo interessati alla stringa di nodi terminali dominata da A; visto che il non-terminale A è quello che si ripete, per ipotesi, almeno due volte nell'albero, ipotizziamo che la stringa dominata dalla A più alta sia vxy e che la A più bassa domini semplicemente x. Chiamiamo u e z le stringhe dominate S rispettivamente alla sinistra e alla destra di x.
Questa la situazione che si viene a creare:



e. Visto che in linea di principio nella derivazione possono comparire un numero ≥ 0 di A, nella stringa compariranno anche un numero ≥ 0 di v e y. La stringa avrà quindi la seguente forma: $\mathbf{uv^ixy^iz}$ con v e y non entrambe nulle e i ≥ 0.

Siamo a questo punto pronti ad enunciare il teorema:
Se L è un linguaggio Context-Free infinito, allora esiste una costante ***k****, tale che qualsiasi stringa* ***w*** *in L più lunga di* ***k*** *può essere scomposta in sottostringhe* $\boldsymbol{w = uv^ixy^iz}$ *con* $i \geq 0$

Per dimostrare che lingue che mostrano fenomeni come il multiple agreement (che prende la forma di un linguaggio $a^nb^nc^n$) o cross-serial dependencies ($a^nn^mc^nd^m$), non sono context-free si procede in modo del tutto simile a quanto mostrato in (26) per il linguaggio a^nb^n.

(37) Anche per le CFG (e quindi per le grammatiche di tipo 2) possiamo fornire, in modo non formale, alcune prove di inadeguatezza per la rappresentazione delle lingue naturali basate sull'**ineleganza** o sull'**implusibilità psicolinguistica**:
a. una prima consiste nell'impossibilità di stabilire direttamente delle **relazioni tra strutture** diverse dal punto di vista "superficiale" ma equivalenti da un punto di vista "profondo". Ad esempio la forma attiva e passiva di una frase devono essere generate da regole diverse
b. problemi simili sono forniti dall'**accordo**. Come nelle grammatiche di tipo 3, anche con le CFG si presenta il problema della crescita esponenziale del numero di regole che occorrono per determinare le corrette giustapposizioni ad esempio tra determinanti e nomi, come mostrato in (34).

(38) **proprietà utili**: oltre alla relazione di precedenza le CFG riescono, contrariamente agli FSA, a determinare relazioni di **dominanza** (e quindi implicitamente di **costituenza**) tra gli elementi linguistici.

Linguaggi di tipo 1 e tipo 0 – Context-Sensitive Grammars e Tree-Adjoining Grammars

(39) I problemi che i linguaggi di tipo 2 si portano dietro sono identificati con la dipendenza cross-seriale, l'accordo multiplo e la reduplicazione. Con i linguaggi di tipo 1 questi fenomeni vengono adeguatamente trattati. Quindi essendo questa classe di linguaggi un insieme proprio dei linguaggi di tipo 0, anche questi ultimi saranno adeguati almeno osservativamente. Il problema che si pone è però a questo punto inverso: le grammatiche in questione risultano fin troppo potenti generativamente e generalizzano strutture "non naturali" per il linguaggio umano come le seguenti:
$\{a^nb^{n!}\}$ oppure $\{a^n\}$ con *n* numero primo

I linguaggi di tipo 2, di tipo 1 e di tipo 0 condividono tutte le proprietà delle Grammatiche a Struttura Sintagmatica e si differenziano semplicemente per le regole di produzione che ammettono: come mostrato in (19) le Grammatiche non ristrette (linguaggi di tipo 0) accettano qualsiasi tipo di regola purché l'antecedente non sia nullo ($\alpha \rightarrow \beta$ (con $\alpha \neq \varepsilon$)).
Le grammatiche di tipo 1 pongono un'ulteriore restrizione: che la lunghezza della forma riscritta sia lunga almeno quanto la forma da riscrivere.

Le CSG non hanno la limitazione delle CFG che ammettevano un solo carattere nella parte sinistra delle regole di produzione. Questo permette di aggirare ad esempio il problema dell'accordo multiplo (che assume una forma del tipo $a^nb^nc^n$) con un limitato numero di regole:
$S \rightarrow aSBC$
$S \rightarrow aBC$
$CB \rightarrow BC$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$

(40) Proprietà computazionali: mentre le grammatiche di tipo 0 richiedono dispositivi estremamente potenti per essere parseggiate (macchine di Turing ideali, cioè senza limitazioni di memoria), le grammatiche Context-Sensitive si accontentano di un Automa Linearmente Limitato (**Linear Bounded Automaton**, cioè una macchina di Turing con un nastro di lunghezza limitata).

(41) All'interno delle context-sensitive grammars, si è quindi cercato di ridurre la potenza generativa in modo da catturare esattamente tutti i possibili fenomeni linguistici senza spreco di risorse computazionali. In questa

ottica vari formalismi vengono postulati. Un'interessante idea è quella delle **grammatiche ad unione di alberi** (**Tree-Adjoining Grammars**, **TAG**, Joshi 75).

I linguaggi generato dalle TAG si dicono **debolmente context-sensitive** perché catturano solo poche proprietà (ma rilevanti) in più delle CFG e risultano propriamente contenute nella classe delle CSG.

Una TAG può essere definita come una quintupla **<Σ, N, I, A, S>** dove:

Σ è un insieme finito di simboli terminali

N è un insieme finito di simboli non terminali ($N \cap \Sigma = \emptyset$)

I è un insieme finito di **alberi iniziali**, caratterizzati come segue:
- i. i nodi interni sono etichettati con simboli non terminali;
- ii. i nodi estremi possono essere etichettati con nodi non terminali (ed in tal caso dovranno sottostare all'operazione di **sostituzione**, marcata convenzionalmente con una freccia verso il basso ↓), o terminali.

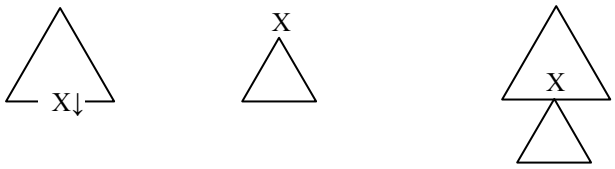
A è un insieme di **alberi ausiliari**, caratterizzati come segue:
- i. i **nodi interni** sono etichettati con simboli non terminali;
- ii. i **nodi estremi** possono essere etichettati con nodi non terminali (ed in tal caso dovranno sottostare all'operazione di **sostituzione**, ad eccezione di un **nodo piede**, marcato convenzionalmente con asterisco *****, il cui simbolo è identico al **nodo radice**), o terminali.

S è il simbolo iniziale della derivazione (Sentence)

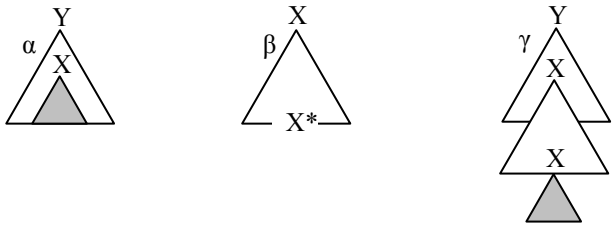
$I \cup A$ fornisce l'insieme degli **alberi elementari**. Un albero composto da più alberi binari combinati insieme secondo le regole che verranno specificate si chiama **albero derivato**.

all'interno delle TAG sono normalmente definite due proprietà (di cui la seconda riducibile alla prima attraverso alcuni accorgimenti):

sostituzione – è l'operazione che permette, all'interno di un albero α di rimpiazzare un nodo non terminale estremo marcato per tale operazione



aggiunzione – è l'operazione che permette di costruire un nuovo albero γ partendo dall'albero ausiliario β, il cui nodo radice è X, e da α (albero iniziale, ausiliario o derivato) che comprende un nodo anch'esso etichettato X, non marcato per la sostituzione:



(42) tra le caratteristiche interessanti e gradite delle TAG vanno sottolineate:
- a. la proprietà di gestire direttamente **oggetti strutturati** e non semplici stringhe
- b. la loro "**moderata**" **context-sensitività**
- c. esistenza di una versione del **pumping lemma per le TAG**
- d. ordine di complessità accettabile in fase di parsing: **$O(n^6)$**

Complessità e psicolinguistica

(43) Dal punto di vista psicolinguistico la **complessità** di una frase viene associata alla **difficoltà** che un parlante incontra nel comprenderla: in questo senso una frase con tre incassamenti centrali è più complessa di una con due, come una che presenta tre dipendenze cross-seriali è più complessa di una con solo due. E' interessante invece notare come non ci sia una grande differenza di complessità in frasi con due, tre o quattro relative non incassate (e venne il cane che morse il gatto che scacciò il topo che mio padre alla fiera comprò) o con due, tre o quattro sintagmi coordinati (Il bel Marco, il buon Mario, la cara Luisa e la triste Maria andarono al mare).

Ricerche sul processamento linguistico umano e sulla complessità percepita tengono conto di questi dati cercando di verificare se la complessità definita in termini formali è in qualche modo correlata con questi fenomeni. In questa direzione è l'osservazione di Pullum e Gazdar (1982) che notano, ad esempio, come il processamento di strutture non context-free causi le difficoltà maggiori.

Purtroppo non esistono dispositivi formali per descrivere una grammatica in cui N incassamenti sono possibili mentre N+1 no. Inoltre sarebbe scorretto dire che le frasi con N+1 incassamenti non sono grammaticali, poiché nessun principio strutturale risulta teoricamente violato. Quindi la complessità percepita sembra collegata non tanto alla struttura grammaticale quanto al **tipo di processamento** richiesto. La generalizzazione sembra inoltre essere valida crosslinguisticamente.

(44) Molti tentativi per spiegare questo fatto fanno ricorso all'idea che la **memoria di processamento** sia **limitata**: da una parte la capacità di tale buffer potrebbe **non** essere **sufficiente** a contenere più di N strutture; dall'altra inserire in questo deposito più di un certo numero di sintagmi (simili?) incompleti potrebbe causare **confusione**.

Yngve 60 propone ad esempio che il processamento linguistico effettivo sia basato su una limitata memoria (limited-size stack) in cui vengono inseriti i risultati parziali delle regole applicate finché i sintagmi non sono completati. Più frasi incomplete vengono inserite nello stack, più difficile da processare sarà il periodo. Questo porta Yngve a concludere che in realtà il linguaggio potrebbe essere generato anche da semplici espressioni regolari, visto che una CFG con **limitazioni sulla quantità di ricorsioni** possibili può essere agilmente modellata attraverso espressioni regolari o FSA.

Gibson (98) propone una teoria diversa, la **Syntactic Prediction Locality Theory** (**SPLT**), con cui predice che il **carico mnemonico totale** richiesto da una struttura sintattica è uguale alla **somma dei carichi mnemonici richiesti dalle singole parole** necessarie per completare tale struttura: in questo senso una frase composta da una serie di sintagmi nominali senza nessun verbo richiederà un certo numero di verbi per essere completata e quindi avrà un livello di complessità maggiore di una semplice serie di frasi dalla struttura SVO (tale ordine vale per l'inglese o per l'italiano); l'introduzione di un pronome che si riferisce ad un'entità già introdotta nell'universo del discorso è ad esempio meno complessa mnemonicamente dell'introduzione di un nuovo referente che dovrà in qualche modo essere integrato nella struttura non ancora completata.

Bibliografia essenziale:

Jurafasky & Martin 2000 *Speech & Language Processing*. Pretience Hall, NJ (**Cap. 13**)
Turcato 1993 *Grammatiche formali e linguaggio naturale*. Calderini Bologna (**Cap. 1, 2**)

Approfondimenti:

- Barton G.E., Berwick R. & Ristad E.S. 1987. *Computational Complexity and Natural Language*. MIT Press
- Chomsky N. 65 *Aspects of the Theory of Syntax*. MIT Press;
- Hopcroft, Motwani & Ullman (2001) *Introduction to the automata theory, languages and computation*. Addison-Wesley. Boston
- Joshi & Schabes (1994) *Tree-Adjoining Grammars*.
- Partee, Meulen & Wall (1993) *Mathematical Method in Linguistics*. Kluwer Academic Publisher. Dordrecht
- Van de Koot H. *The Computational Complexity of natural language recognition*. Ms. University College London