

IUSS - Pavia  
A.A.2016/17  
CRISTIANO CHESI

PhD program in  
Cognitive Neurosciences and Philosophy of Mind (cycle XXXII)

## INTRODUCTION TO LINGUISTIC COMPUTATION AND ON-LINE PROCESSING OF TOUGH SENTENCES

## Index

- ⊙ **Grammatiche formali**
  - Approcci formali alla **competenza linguistica**
  - Grammatiche a struttura sintagmatica (PSG) e la **gerarchia di Chomsky**
- ⊙ **Teoria della computazione linguistica**
  - Macchine di Turing (universali) e altri formalismi
  - Introduzione alla **computabilità, complessità e automata theory**.
  - **Parsing** e complessità
- ⊙ **Approfondimenti su grammatiche formali e processing**
  - **Competence & Performance**
  - Derivazioni **Minimali(ste)**
  - Complessità in termini di **intervenienza**

Intro to linguistic computation

C. Chesi

## Letture dell'Antologia Generale

- ⊙ Chesi (2015) [Il processamento in tempo reale delle frasi complesse](#). In atti del convegno "Compter Parler Soigner", E.M. Ponti (ed). Pavia University Press.
- ⊙ Partee B., A. ter Meulen & R. Wall (1990) *Mathematical Methods in Linguistics*, Springer, 1990, **solo Capitoli 16 – 18 (pagg. 433-506)**
- ⊙ Lazzari M., A. Bianchi, M. Cadei, C. Chesi e S. Maffei (2010) [Informatica umanistica](#), McGraw-Hill Education, 2010, **solo Capitolo 4 (pagg. 147-203)**
- ⊙ Graffi G. & S. Scalise (2003) *Le lingue e il linguaggio. Introduzione alla linguistica*, Il Mulino
- ⊙ Donati C. (2008) *La sintassi. Regole e strutture*, Il Mulino

Intro to linguistic computation

C. Chesi

## Altre letture di approfondimento

- ⊙ Baddeley A., M. W. Eysenck, M. Anderson (2011) *La Memoria*. Il Mulino
- ⊙ Chesi C., A. Moro (2014) Computational complexity in the brain. In Frederick J. Newmeyer and Laurel B. Preston (eds.), *Measuring Linguistic Complexity*. Oxford: Oxford University Press
- ⊙ Chesi C. (2014) On directionality of phrase structure building. *Journal of Psycholinguistic Research*. 44(1) <http://dx.doi.org/10.1007/s10936-014-9330-6>
- ⊙ Hopcroft, Motwani & Ullman (2001) *Introduction to the automata theory, languages and computation*. Addison-Wesley. Boston
- ⊙ Jurafsky, D. & Martin, J. H. (2000) *Speech and Language Processing*. Prentice-Hall.
- ⊙ Stabler, E. 1997. Derivational minimalism. in Retoré, ed. *Logical Aspects of Computational Linguistics*. Springer
- ⊙ Sprouse, J., Wagers, M., & Phillips, C. (2012). Working-memory capacity and island effects: A reminder of the issues and the facts. *Language*, 88
- ⊙ Van Dyke, J. A., & McElree, B. (2006) Retrieval interference in sentence comprehension. *Journal of Memory and Language*, 55(2), 157-166.

Intro to linguistic computation

C. Chesi

I puntata

## LE GRAMMATICHE FORMALI

## La competenza linguistica

- ⊙ Conoscere una lingua come l'Italiano significa sapere che:
  - una parola può iniziare per *ma...* (*mare*) ma non per *mr...*
  - la *e* di *case* ha un valore diverso da quella di *mare*
  - "le case sono sulla collina" Vs. "\*case le collina sono sulla"
  - "il gatto morde il cane" >  
**sogg:** gatto (**agente**); **predicato:** morde (**evento**); **ogg:** cane (**paziente**)
  - ?il tostapane morde il gatto
  - l'espressione "queste case" si riferisce ad un gruppo preciso di case, evidente dal contesto (Vs. "delle case")
- ⊙ La **competenza linguistica** è una conoscenza (**finita**) che ci permette di:
  - Sapere quali (**infinite**) espressioni fanno parte della lingua in questione
  - Assegnare a queste espressioni (almeno) un **significato**

Intro to linguistic computation

C. Chesì

## La competenza linguistica

- ⊙ Tracciare un confine che includa le cose che vogliamo spiegare:
  - **Ordine** delle parole > significato  
es. ho visto un uomo nel parco con il cannocchiale
  - **Accordo**  
es. \*la mela rosso  
\*Gianni corrono  
Gianni ha visto Maria vs. Gianni l'ha vista (Maria)
  - **Dipendenze a distanza** (legamento pronominale, movimento sintattico)  
es. cosa credi che Maria abbia chiesto a Luigi di comprare \_?  
Gianni<sub>i</sub> promette a Maria<sub>j</sub> di \_<sub>i</sub> andare a trovarla<sub>j/k</sub>  
Gianni<sub>i</sub> chiede a Maria<sub>j</sub> di \_<sub>i</sub> andare a trovarla<sub>j/k</sub>

Intro to linguistic computation

C. Chesì

## Livelli di adeguatezza di una grammatica

- ⊙ **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:
  - **osservativa:** la lingua definita dalla grammatica coincide con quella che si intende descrivere
  - **descrittiva:** l'analisi grammaticale proposta è in linea con le intuizioni linguistiche dei parlanti fornendo descrizioni strutturali adeguate delle frasi accettabili
  - **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.

Intro to linguistic computation

C. Chesì

## Nozioni formali di base

- ◉ **Insiemi definiti estensivamente** (finiti):  $A = \{a, b, c\}$
- ◉ **Insiemi definiti induttivamente** (infiniti):  $A = \{x: x \text{ ha la proprietà } p\}$
- ◉ **Insiemi ordinati** ( $n$ -uple):  $A = (a, b, c)$
- ◉ **Cardinalità:**  $|A| = \text{numero di elementi di } A$
- ◉ **Prodotto cartesiano:**  $A = \{a, b, c\}$        $B = \{x, y\}$   
 $A \times B = \{(a, x), (b, x), (c, x), (a, y), (b, y), (c, y)\}$
- ◉ **Unione:**  $A \cup B = \{x: x \in A \text{ oppure } x \in B\}$
- ◉ **Concatenazione:**  $A \circ B = \{xy: x \in A \text{ e } y \in B\}$
- ◉ **Star:**  $A^* = \{x_1 x_2 \dots x_n: n \geq 0 \text{ e ogni } x_i \in A\}$

Intro to linguistic computation

C. Chesì


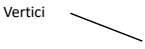
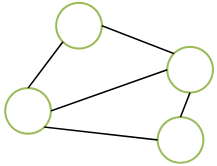
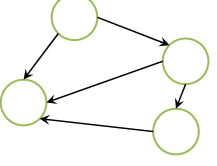
## Nozioni formali di base

- ◉ **Indici:**  $x_k = k$ -esimo elemento di una serie  
 $x^k = \text{una serie di } k \text{ elementi}$   
 $X^R = \text{l'immagine speculare di } X$
- ◉ **Funzione:**  $f(x) \rightarrow y$       ( $x = \text{Domain}, y = \text{Range}$ ):
- ◉ **Predicati:**  $f(x) \rightarrow \{\text{vero}, \text{falso}\}$
- ◉ **Predicati n-ari:**  $f(x, y \dots z) \rightarrow \{\text{vero}, \text{falso}\}$
- ◉ **Relazioni di equivalenza:** predicati binari  $R$  per cui valgono le seguenti proprietà:
  - $R$  è **riflessiva**, cioè per ogni  $x, xRx$ ;
  - $R$  è **simmetrica**, cioè per ogni  $x$  e  $y$ , se  $xRy$  allora  $yRx$ ;
  - $R$  è **transitiva**, cioè per ogni  $x, y$  e  $z$ , se  $xRy$  e  $yRz$  allora  $xRz$ ;

Intro to linguistic computation

C. Chesì

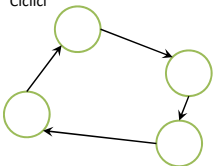
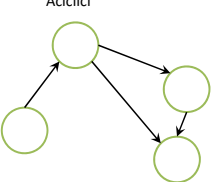
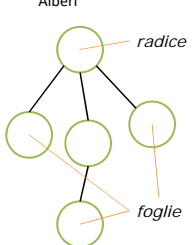
## Nozioni formali di base

- ◉ **Grafi**
- Nodi 
- Vertici 
- Non direzionali 
- Direzionali 

Intro to linguistic computation

C. Chesì

## Nozioni formali di base

- ◉ **Grafi**
- Ciclici 
- Aciclici 
- Alberi 
- Grado:** numero di vertici che esce/entra da un nodo

Intro to linguistic computation

C. Chesì

## Come si formalizza una grammatica

- ⊙ **A = Alfabeto**  
insieme finito di caratteri ( $A^*$  = l'insieme di tutte le stringhe possibili costruite concatenando elementi di A;  $\epsilon$  è l'elemento nullo)
- ⊙ **V = Vocabolario**  
insieme (potenzialmente infinito) di parole, costruite concatenando elementi di A ( $V \subseteq A^*$ )
- ⊙ **L = Linguaggio**  
insieme (potenzialmente infinito) di frasi, costruite concatenando elementi di V ( $L \subseteq V^*$ )

Intro to linguistic computation

C. Chesì

## Come si formalizza una grammatica

- ⊙ Una **grammatica formale** per il linguaggio L è un insieme di regole che permettono di **generare/riconoscere** tutte e sole le frasi appartenenti a L e (d eventualmente) di assegnare a queste frasi un'adeguata descrizione strutturale.
- ⊙ Una grammatica formale G deve essere:
  - **esplicita** (il giudizio di grammaticalità deve essere frutto solo dell'applicazione meccanica delle regole scelte)
  - **consistente** (una stessa frase non può risultare allo stesso tempo grammaticale e non grammaticale)

Intro to linguistic computation

C. Chesì

## Come si formalizza una grammatica

- ⊙ Una grammatica formale G può essere formalizzata (grammatica a struttura sintagmatica o **Phrase Structure Grammar, PSG** Chomsky 1965), come una quadrupla ordinata  $(V_N, V_T, \rightarrow, \{S\})$  dove:
  - $V_N$  è l'insieme di tutte le categorie non terminali utilizzabili nelle descrizioni strutturali
  - $V_T$  è il vocabolario della lingua che racchiude tutti e soli gli elementi terminali ( $V_T \cup V_N = V$ )
  - $\rightarrow$  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  $A \in V_N$   $\phi A \psi \rightarrow \phi \tau \psi$  per qualche  $\phi, \tau, \psi \in V^*$
  - $\{S\}$  è un sottoinsieme di  $V_N$  definito come l'insieme degli assiomi che convenzionalmente contiene il solo simbolo S.

Intro to linguistic computation

C. Chesì

## Come si formalizza una grammatica

- ⊙ Date due stringhe  $\phi$  e  $\psi \in V^*$  si dice che esiste una  **$\phi$ -derivazione di  $\psi$**  se  $\phi \rightarrow^* \psi$ .
- ⊙ Se esiste una  $\phi$ -derivazione di  $\psi$  allora si può anche dire che  **$\phi$  domina  $\psi$** . Tale relazione è riflessiva e transitiva.
- ⊙ Una  $\phi$ -derivazione di  $\psi$  si dice **terminata** se:
  - $\psi \in V_T^*$
  - per nessun  $\chi$  esiste una  $\psi$ -derivazione di  $\chi$
- ⊙ Data una grammatica G, una lingua generata da G, detta **L(G)**, è l'insieme di tutte le stringhe  $\phi$  per cui esiste una S-derivazione terminata di  $\phi$ .

Intro to linguistic computation

C. Chesì

## Descrizioni strutturali (cioè alberi sintattici)

- ⊙ Una **Descrizione strutturale** è una quintupla **(V, I, D, P, A)** tale che:
  - V** è un insieme finito dei vertici (es.  $v_1, v_2, v_3, \dots$ )
  - 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, irreflessiva, antisimmetrica e transitiva) definita su V
  - A** è la funzione di assegnazione; una funzione non suriettiva da V a I

Intro to linguistic computation

C. Chesì

## Capacità generativa e relazioni di equivalenza

- ⊙ La capacità generativa denota l'insieme di oggetti generati dalla grammatica; tale capacità è:
  - **debole** se riferita al solo semplice insieme di frasi generabili
  - **forte** se associa a tali frasi l'appropriata descrizione strutturale
- ⊙ Due grammatiche si dicono **equivalenti** se sono in grado di **generare** lo stesso insieme di oggetti.  
Anche qui si può parlare di **equivalenza debole** o **equivalenza forte**

Intro to linguistic computation

C. Chesì

## Decidibilità

- ⊙ Un insieme  $\Sigma$  si dice
  - **decidibile** (o ricorsivo) se per ogni elemento  $e$  appartenente all'insieme universo esiste un **procedimento meccanico** M che permette di stabilire in un **numero finito di passi** se  $e \in \Sigma$  oppure  $e \notin \Sigma$  (la non appartenenza a  $\Sigma$  determina l'appartenenza al complemento di  $\Sigma$  definito come  $\bar{\Sigma}$ )
  - **ricorsivamente enumerabile** quando esiste un procedura che enumera tutti e soli gli elementi di  $\Sigma$

Intro to linguistic computation

C. Chesì

## Grammatiche Regolari – Linguaggi Regolari

- ⊙ Le grammatiche regolari sono grammatiche che ammettono solo regole sistematicamente di questo tipo:

$A \rightarrow xB$

oppure (sistematicamente) di questo:

$A \rightarrow Bx$

I linguaggi generati da queste grammatiche si definiscono **Regolari**

Intro to linguistic computation

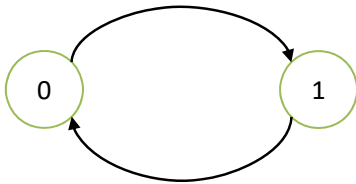
C. Chesì

## Automi e computazione

◉ Gli **automi** sono modelli matematici che permettono di definire una computazione descrivendo relazioni tra «stati» di un sistema.

◉ Ecco un esempio di automa (a stati finiti): l'interruttore!

- 0 = acceso
- 1 = spento
- -> = pressione



Intro to linguistic computation

C. Chesì

## Macchine e Stati Finiti (FSA)

◉ **Finite-State Automata (FSA)**

definiti come quintuple  $\langle Q, \Sigma, q_0, F, \delta \rangle$  dove:

$Q$  = insieme finito e non nullo di stati

$\Sigma$  = alfabeto finito e non nullo di caratteri accettabili in input

$q_0$  = stato iniziale, con  $q_0 \in Q$

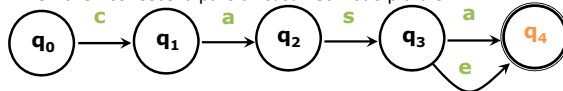
$F$  = insieme di stati finali, con  $F \subseteq Q$

$\delta$  = insieme delle regole di transizione definite in  $Q \times \Sigma$  su  $Q$

## FSA usati come riconoscitori di parole

◉ un insieme di FSA non è solo un insieme di macchine che permettono di **riconoscere** o **rifiutare** un elemento lessicale, ma anche di **rappresentare** una frase (o tutta una lingua...).

◉ FSA che riconosce la parola *casa* ed il suo plurale:



$Q = \{q_0, q_1, q_2, q_3, q_4\}$ ,

$\Sigma = \{c, a, s, e, \#\}$ ,

$Q_0 = \{q_0\}$ ,

$F = \{q_4\}$ ,

$\delta =$

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
$c$	$q_1$				
$a$		$q_2$			$q_4$
$s$			$q_3$		
$e$					$q_4$

## Le Espressioni Regolari

◉ Le **Espressioni Regolari** sono notazioni algebriche 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.

Espressione Regolare	Corrispondenza	Es. pattern identificato
$[Dd]uomo$	<u>Duomo</u> oppure <u>duomo</u>	Il <u>duomo</u> è nella piazza
$[^a-z]$	tutto fuorché lettere minuscole	Il duomo è ...
$sal?ta$	<u>salita</u> oppure <u>salta</u>	Marco deve <u>saltare</u>
$sal.ta$	accetta ogni carattere tra le i e la t	Marco saluta
$bu^*$	b seguito da un numero imprecisato (anche nullo) di u	buuuuu! oppure b!
$^L \quad Vs. \quad a\$$	$^$ = inizio stringa; $\$$ = fine stringa	<u>La casa</u>
$cas(a e)$	è equivalente alla disgiunzione logica	Marco vive in un casale
$\backslash^*$	il backslash è il simbolo di escape	A*

## RG, FSA e RE sono equivalenti

- ◉ **Grammatiche Regolari (RG, Regular Grammar)**, le **Macchine a Stati Finiti (FSA, Finite State Automata)** le **Espressioni Regolari (RE, Regular Expressions)** sono equivalenti (i.e. descrivono lo stesso insieme di linguaggi: i linguaggi regolari).

Dimostrazione per costruzione

Intro to linguistic computation

C. Chesì

## Digressione: Espressioni Regolari ed Eliza

- ◉ **Eliza**, un programma del 1966 scritto da Joseph Weizenbaum. Eliza simula il comportamento di uno psicoterapeuta Rogeriano che intraprende una conversazione con un paziente:

**paziente:** gli uomini sono tutti uguali  
**eliza:** in che senso  
**paziente:** ci fregano sempre in qualche modo  
**eliza:** puoi fare un esempio specifico  
**paziente:** un mio amico mi ha fatto venire qui  
**eliza:** un tuo amico ti ha fatto venire qui  
**paziente:** dice che sono quasi sempre depressa  
**eliza:** mi dispiace sentire che sei depressa  
**paziente:** è vero sono infelice  
**eliza:** credi che venire qui ti aiuterà a non essere infelice  
**paziente:** ho bisogno di aiuto, questo è sicuro  
**eliza:** cosa intendi come aiuto  
**paziente:** forse potrei riuscire ad andare d'accordo con mia madre  
**eliza:** parlami di tua madre

## Digressione: Espressioni regolari ed Eliza

- ◉ **Espressioni Regolari e l'operazione di Sostituzione**
  - 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:
    - $s/\text{espressione\_regolare1}/\text{espressione\_regolare2}/$
    - $s/\text{non è } [a-z]^* ?[a-z]^* \text{ interessante} / \text{è assolutamente interessante}/$
- ◉ **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:
  - $s/ \text{il } (\text{libro} | \text{gelato}) \text{ è stato comprato da } (\text{Maria} | \text{Gianni}) / \backslash 2 \text{ ha comprato il } \backslash 1 /$  permette di costruire la forma attiva (**Gianni** ha comprato il **libro**) della frase passiva (il **libro** è stata comprata da **Gianni**).
- ◉ operazioni di **sostituzione** in ELIZA:
  - $s/ \text{sono } [.* ] ](\text{depress}[\text{o} | \text{a}] | \text{triste})/\text{sono spiacente di sapere che sei } \backslash 1 /$
  - $s/ \text{sono tutt}[\text{i} | \text{e}] (.* ) / \text{in che senso sono } \backslash 1 ? /$
  - $s/ \text{sempre} / \text{potresti far riferimento ad un esempio specifico?}$

Intro to linguistic computation

C. Chesì

## Digressione: Espressioni regolari e ...



Siri.  
Ogni tuo desiderio  
è un ordine.

Con Siri basta usare la voce per inviare messaggi, cercare ritorni, telefonare e fare molte altre cose. Fai una domanda parlando con naturalezza. Siri capisce e non dice, sa cosa intendi, e soprattutto ti risponde. È così utile e facile da usare che finirà per diventarti indispensabile.



Google Now. Le informazioni  
giuste al momento giusto.

Schede utili con le informazioni che ti servono nel corso della giornata, visualizzate ancor prima che tu le chieda.

Intro to linguistic computation

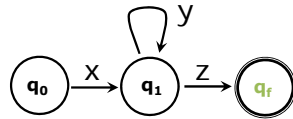
C. Chesì

## Il linguaggio naturale è generabile dalle Grammatiche Regolari?

### ⊙ Pumping lemma per le grammatiche regolari

Se  $A$  è un Linguaggio Regolare, allora  $c$ 'è un numero  $p$  (che esprime l'ampiezza del «pompaggio»), per cui, se  $s$  è una qualsiasi stringa di  $A$  di lunghezza almeno equivalente a  $p$ , allora può essere divisa in 3 parti,  $s = xyz$  tali che:

- I. Per ogni  $i \geq 0$ ,  $xy^iz \in A$
- II.  $|y| > 0$
- III.  $|xy| \leq p$



### ⊙ Proprietà di linguaggi non riconoscibili dalle grammatiche regolari

$a^n b^n$  non è una stringa generabile da nessuna grammatica regolare (poiché nessuna sottostringa può essere «pompata» quante volte si vuole garantendo lo stesso numero di  $a$  e di  $b$ )

Intro to linguistic computation

C. Chesì

## Grammatiche Context-Free

- ⊙ Le grammatiche Context-Free (CFG) sono grammatiche che ammettono solo regole sistematicamente di questo tipo:

$A \rightarrow \gamma$  (dove  $\gamma$  è una sequenza qualsiasi di simboli terminali o non terminali)

I linguaggi generati da queste grammatiche si definiscono **Context-Free**

- ⊙ Ogni grammatica CF può essere «convertita» in una grammatica (debolmente) equivalente nella forma chiamata **Chomsky Normal Form (CNF)**:

$A \rightarrow BC$   
 $A \rightarrow a$

Intro to linguistic computation

C. Chesì

## Push-Down Automata

- ⊙ Un **Push-Down Automata (PDA)** è una macchina a stati finiti dotata di uno **stack di memoria**; i PDA sono definiti come sestuple  $\langle Q, \Sigma, q_0, F, \delta, \Gamma \rangle$  dove:

$Q$  = insieme finito e non nullo di stati

$\Sigma$  = alfabeto finito e non nullo di caratteri accettabili in input

$q_0$  = stato iniziale, con  $q_0 \in Q$

$F$  = insieme di stati finali, con  $F \in Q$

$\delta$  = insieme delle regole di transizione definite in  $Q \times \Sigma \times \Gamma$  su  $Q \times \Gamma$

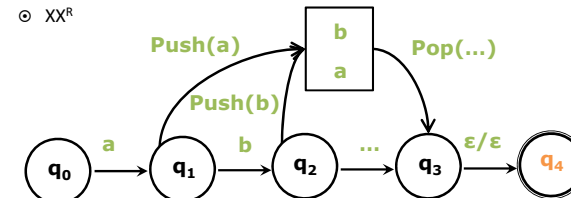
$\Gamma$  = alfabeto di memoria

Intro to linguistic computation

C. Chesì

## PDA usati come riconoscitori di frasi

- ⊙  $XX^R$



$Q = \{q_0, q_1, q_2, q_3, q_4\}$ ,  
 $\Sigma / \Gamma = \{a, b, \epsilon\}$ ,  
 $Q_0 = \{q_0\}$ ,  
 $F = \{q_4\}$ ,  
 $\delta =$

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
$a$	$q_1$ <i>push(a)</i>				
$b$		$q_2$ <i>push(b)</i>			
$\dots$			$q_3$ <i>pop(...)</i>		
$\epsilon$					$q_4$



## CFG e PDA sono equivalenti

- Le **Grammatiche Context-Free** (CFG, Context-Free Grammars), e i **Push-Down Automata** (PDA) sono equivalenti (i.e. descrivono lo stesso insieme di linguaggi: i linguaggi Context-Free).

«Dimostrazione» per costruzione:

- PDA contiene una regola tale che:  $(q_0, \epsilon, \epsilon) \rightarrow (q_1, S)$
- Per ogni regola della CFG tale che  $A \rightarrow x$ , il PDA contiene un'istruzione di questo tipo:  $(q_1, \epsilon, A) \rightarrow (q_1, x)$
- Per ogni simbolo  $a : a \in V_T$ , il PDA contiene un'istruzione del tipo  $(q_1, a, a) \rightarrow (q_1, \epsilon)$

Intro to linguistic computation

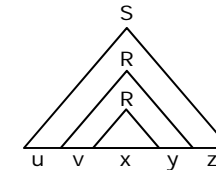
C. Chesì

## Ogni espressione linguistica può essere generata dalle Grammatiche CF?

- Pumping lemma per le grammatiche context-free**

Se  $A$  è un Linguaggio Context-Free, allora c'è un numero  $p$  (che esprime l'ampiezza del «pompaggio»), per cui, se  $s$  è una qualsiasi stringa di  $A$  di lunghezza almeno equivalente a  $p$ , allora può essere divisa in 5 parti,  $s = uvxyz$  tali che:

- Per ogni  $i \geq 0$ ,  $uv^ixy^iz \in A$
- $|vy| > 0$
- $|vxy| \leq p$



- Proprietà di linguaggi non riconoscibili dalle grammatiche regolari**  
 $a^n b^n c^n$  non è una stringa generabile da nessuna grammatica context-free (poiché nessuna tripla di sottostringhe può essere «pompata» indefinitivamente garantendo lo stesso numero di  $a$  di  $b$  e di  $c$ )

C. Chesì

## Inclusioni tra classi di grammatiche

- La **gerarchia di Chomsky** (1956, 59):

**Tipo 3:** grammatiche regolari (**Regular Grammars**):  
 $A \rightarrow xB$  (eq. Finite State Automata)

**Tipo 2:** grammatiche non-contestuali (**Context Free Grammars**):  
 $A \rightarrow \gamma$  (eq. Push-Down Automata)

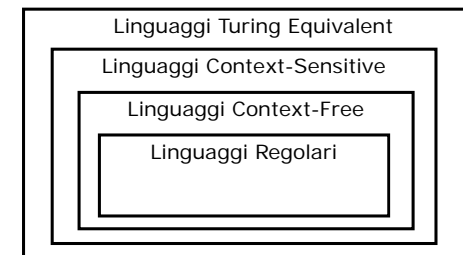
**Tipo 1:** grammatiche contestuali (**Context Sensitive Grammars**):  
 $\alpha A \beta \rightarrow \alpha \gamma \beta$  ( $\gamma \neq \epsilon$ ) (eq. Linear-Bounded Automata)

**Tipo 0:** grammatiche non ristrette (**Turing Equivalent Grammars**):  
 $\alpha \rightarrow \beta$  ( $\alpha \neq \epsilon$ ) (eq. Augmented Transition Networks)

Intro to linguistic computation

C. Chesì

## Inclusioni tra classi di grammatiche



Intro to linguistic computation

C. Chesì

## Dove stanno le lingue naturali?

- Le lingue naturali **non** sono **generabili** da **grammatiche regolari** (Chomsky 1956):

**If A then B** (con A e B potenzialmente anch'esse nella forma "if X then Y"... quindi linguaggi di tipo  $a^n b^n$ )

- Le lingue naturali **non** sono **generabili** da **grammatiche context-free** (Shieber 1985):

Jan säit das mer em Hans es huus hälfed aastriche  
("famoso" dialetto svizzero tedesco)  
J. dice che noi a H. la casa abbiamo aiutato a dipingere

Gianni, Luisa e Mario sono rispettivamente  
sposato, divorziata e scapolo  
( "ABC...ABC"... quindi linguaggi di tipo XX)

Intro to linguistic computation

C. Chesì

## Dove stanno le lingue naturali?

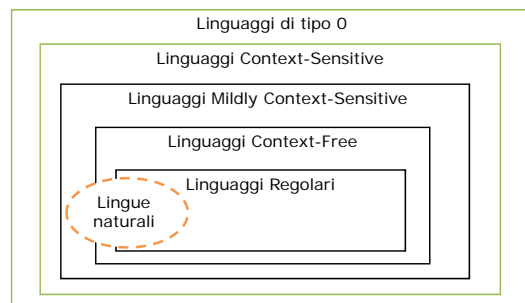
- Ricorsività** nelle lingue naturali, ovvero come fare un uso infinito di mezzi finiti:

- Incassamento a destra** ( $ab^n$ : iterazione):  
[il cane morse [il gatto [che rincorse [il topo [che scappò]]]]]]
- Incassamento centrale** ( $a^n b^n$ : counting recursion):  
[il topo [che il gatto [che il cane morse] rincorse] scappò]
- Dipendenze cross-seriali** (xx, identity recursion)  
Gianni, Maria e Marco sono rispettivamente sposato, nubile e divorziato

Intro to linguistic computation

C. Chesì

## Dove stanno le lingue naturali?



Intro to linguistic computation

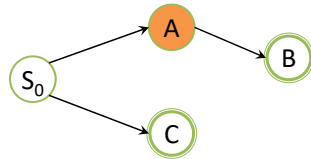
C. Chesì

Il puntata

## TEORIA DELLA COMPUTAZIONE (LINGUISTICA)

## L'utilità di un modello computazionale

- Prevedere possibili disfunzioni



- Calcolare la complessità di certi processi...

Intro to linguistic computation

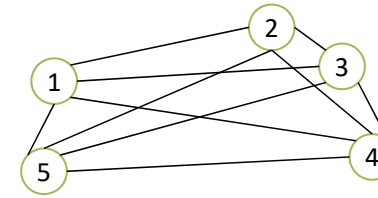
C. Chesì

## Cos'è la complessità

- Ordinate 5 numeri (**sorting problem**)



- Trovate il percorso più breve che collega 5 città (**travelling salesman problem**)



Intro to linguistic computation

C. Chesì

## Cos'è computabile

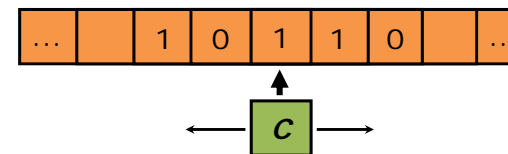
- (informalmente) per **computazione** si intende la specificazione della **relazione tra un input ed un output**. Questa relazione può essere definita in vari modi formalmente chiamati **algoritmi**: descrizioni di una **serie di stadi e transizioni** tra questi stati in cui l'informazione di input viene **trasformata** prima di raggiungere la forma dell'output. Una computazione tenta quindi di ridurre ogni input ad output seguendo una serie di **passi consentiti** dal modello computazionale.
- La tesi di Turing-Church** (semplificata) ogni computazione che può essere realizzata da un qualsiasi **dispositivo fisico** può essere realizzata anche da un **algoritmo**; se il dispositivo fisico riesce a completare il compito in  $n$  passi, tale algoritmo potrà riuscirci in  $m$  passi, **con  $m$  che differisce da  $n$  di al massimo un polinomiale**.
- Alcuni algoritmi impiegano **troppo tempo** per fornire una soluzione
- Altri algoritmi possono **non terminare mai**

Intro to linguistic computation

C. Chesì

## Macchine di Turing

- nastro infinito diviso in celle
- alfabeto  $A$  (di almeno 2 elementi, ad esempio  $A = \{0, 1\}$ )



- cursore  $C$  (che scorre a sinistra e a destra, che può leggere, cancellare e scrivere un carattere)
- insieme finito  $Q$  di stati  $(q_0, q_1, \dots, q_n)$
- input finito  $I$  costituito da caratteri in  $A$
- insieme finito  $S$  di stati della macchina descritte da quintuple del tipo  $\langle q, a, v, q' \rangle$  dove  $q, q' \in Q$ ;  $a, b \in A$ ;  $v = \{\text{destra, sinistra}\}$

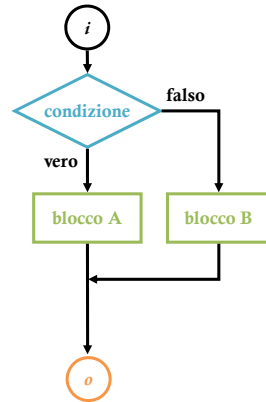
Intro to linguistic computation

C. Chesì

## Flow charts

⊙ Grafo orientato etichettato **costituito da:**

- un **ingresso** (dove viene introdotto l'input  $i$ )
- una o più **uscite** (da cui viene, eventualmente, recuperato un output  $o$ )
- un insieme finito di **blocchi di istruzioni** tali che ogni istruzione è del tipo  $X = Y$ ,  $X = X+1$ ,  $X = X-1$
- un insieme finito (possibilmente nullo) di blocchi speciali, detti **condizioni** tali da presentare interrogazioni booleane del tipo  $X = Y?$
- un insieme finito di **connettori** che collegano i blocchi, tali che da ogni blocco esce 1 ed 1 sola freccia, mentre dal blocco condizionale escono 2 frecce



Intro to linguistic computation

C. Chesì

## Modularità

- ⊙ **Macchine di Turing e flow charts** sono equivalenti: esprimono la stessa classe di funzioni: le funzioni computabili.
- ⊙ Entrambi i formalismi godono della proprietà della **composizionalità** (M1 • M2). Questo ci garantisce che un algoritmo può in realtà essere il risultato di una composizione di più macchine di Turing (o di più flow charts).
- ⊙ Si dice “divide et impera” il paradigma di programmazione che suggerisce di **scomporre un problema nelle sue sottoparti** prima di iniziare a pensare agli algoritmi che lo risolvono.

Intro to linguistic computation

C. Chesì

## Nozioni di computazione

- ⊙ Per **computazione** si intende la **specificazione della relazione** tra un **input** ed un **output**
- ⊙ tale relazione consiste in una serie di **stadi intermedi** in cui l'informazione di input può venire trasformata prima di raggiungere lo “status” di output
- ⊙ **obiettivo della computazione**: ridurre ogni input ricevuto ad output significativo seguendo una serie di passi consentiti dal modello computazionale adottato
- ⊙ **spazio del problema**: serie di **stati raggiungibili** applicando correttamente le regole ammissibili, dati:
  - uno **stato iniziale** del sistema
  - un preciso **input**

## Nozioni di complessità

- ⊙ La **complessità** di una computazione è direttamente proporzionale alla **quantità di risorse impiegate** per mappare un input in un output.
- ⊙ Tali risorse sono solitamente definite su due dimensioni:
  - **tempo** (time complexity): passi elementari da compiere
  - **memoria** (space complexity): quantità di informazione massima richiesta ad ogni passo
- ⊙ La complessità è **direttamente proporzionale alla dimensione del problema** (ad es. ordinare 1000 parole sarà più complesso che ordinarne 10);
- ⊙ si può intuire che la complessità di una grammatica sarà direttamente proporzionale alla sua **forza generativa**.

## Nozioni di complessità

- La **dimensione del problema** viene espressa in funzione dell'input da processare
- quindi l'**ordine di complessità** si potrà esprimere in funzione di tale input:

$c \cdot n^2$  (problema con ordine di complessità temporale polinomiale)  
 $n$  = dimensione dell'input  
 $c$  = costante data (dipendente dal tipo di computazione)

Si dirà che tale problema ha ordine di complessità pari a  $n^2$  poiché la costante  $c$  sarà irrilevante sulla crescita della funzione per  $n$  che tende all'infinito. Tale ordine di complessità si definisce:  $O(n^2)$ .

## Nozioni di complessità

- interesse nel **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** (uso di risorse sicuramente finite) il calcolo della complessità è relativamente rilevante
- Al crescere dell'input a piacimento ( $n$  che tende all'infinito), come nel caso di qualsiasi grammatica formale linguisticamente interessante, il **tasso di crescita della funzione** è cruciale per determinarne la **trattabilità**
- si dice che un problema è **trattabile** se esiste una procedura che fornisce una soluzione (positiva o negativa) in un **tempo finito**
- Un problema si dice **umanamente trattabile** se l'algoritmo fornisce una risposta in tempi accettabili

## Nozioni di complessità

- 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 alcune funzioni:

dimensione input →	10	20	50	100
↓ funzione				
$N^2$	0,0001 secondo	0,0004 sec.	0,0025 sec.	0,01 sec
$N^5$	0,1 sec.	3,2 sec.	5 minuti e 2 sec.	2 ore e 8 min.
$2^N$	0,001 sec.	1 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

## Problemi classici e calcolo della complessità

- 3SAT problem** (variante del **problema del soddisfacimento**, **satisfiability problem** o **SAT**)  
trovare, se 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:  

$$(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee b \vee c) \wedge \dots$$
- nel **peggiore dei casi**, si dovrebbero provare tutte le combinazioni di assegnazioni possibili, ovvero  $2^N$  (dove  $2$  è il numero dei possibili valori dei singoli proposizionali, Vero e Falso, e  $N$  è il numero dei proposizionali  $a, b, c, \dots$ ).
- 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**.

## Problemi classici e calcolo della complessità

- **QBF problem**  
 (problema della **Quantificazione di Formule Booleane**)  
 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:
   
  

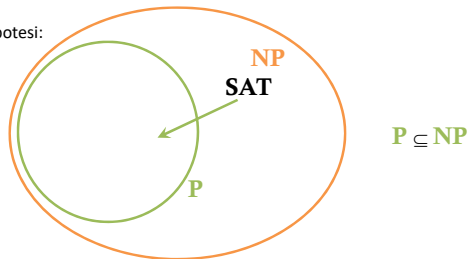
$$Qx_1, Qx_2, \dots, Qx_n F(x_1, x_2, \dots, x_n)$$
 (con Q uguale a  $\forall$  oppure  $\exists$ )
- Il problema è **difficile** da risolvere, **come 3SAT**, ma anche **difficile da verificare**: in effetti, ogni proposizionale di un problema di 3SAT potrebbe essere considerato come **quantificato 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

## Problemi classici e calcolo della complessità

- Ipotizzando che un computer **riesca a risolvere effettivamente** un problema come il 3SAT, tale computer dovrà utilizzare un **algoritmo al peggio polinomiale**.
- vista la natura del problema, tale algoritmo dovrà essere **necessariamente non-deterministico** (quindi implementabile in quella che viene chiamata una **Macchina di Turing (MT) non-deterministica**).
- Questi problemi vengono definiti di tipo **NP**:  
**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 probabilmente inclusa in quella dei problemi in **NP**.
- Per quanto riguarda i problemi (come 3SAT) in NP non esiste prova concreta della loro **riducibilità** a problemi di ordine P.

## Problemi classici e calcolo della complessità

- ipotesi:



- problemi come il **SAT** si dice **NP-hard** (stesse difficoltà dei problemi della classe NP). Ipotizzando un ipotetico dispositivo "preveggennte" in grado di 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.

## Cos'è il parsing

- data una grammatica **G** ed un input **i** fare parsing significa applicare una funzione **p(G, i)** in grado di:
  1. accettare/rifiutare **i**
  2. assegnare ad **i** una struttura descrittiva adeguata (es. Indicatore Sintagmatico) (ad es. il **parsing sintattico** riconosce un input grammaticale e assegna ad esso una struttura sintattica)

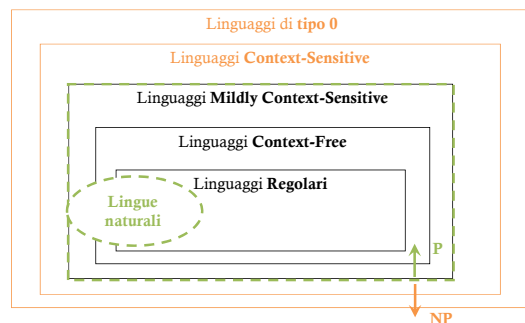
## Universal Recognition Problem (URP) e riduzione

- 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  $x$ ,  $x$  appartiene al linguaggio generabile da  $G$ ?*

## Universal Recognition Problem (URP) e riduzione

- L'**URP** è un problema di parsing generalizzato che può essere ridotto ad un problema come **SAT**
- intuizione di fondo: per una stringa  $x$ , come un proposizionale  $a$  in una formula di **SAT**, 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  $x$  (tutte le occorrenze di  $x$  devono accordarsi / essere compatibili in senso linguistico) che accordo tra i proposizionali in una formula SAT (inteso come consistenza dell'assegnazione di valori). Si deduce quindi che l'URP è almeno complesso quanto il SAT e che è **NP-completo**

## Gerarchia di Chomsky e ordini di complessità



## Complessità psicolinguistica

- complessità = difficoltà** incontrata da un parlante nel comprendere/produrre una frase (es. 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. Mentre non c'è 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")
- ipotesi 1:** complessità formale = complessità psicolinguistica
- ipotesi 2:** memoria di processamento 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**.

## Complessità psicolinguistica

- ⊙ **ipotesi 1**
  - il processamento di **strutture non context-free** causa le difficoltà maggiori (Pullum e Gazdar 1982)
- ⊙ **ipotesi 2**
  - **memoria limitata (limited-size stack, Yngve 1960)**  
il processamento linguistico effettivo è basato su una memoria limitata (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.
  - **Syntactic Prediction Locality Theory (SPLT, Gibson 1998) carico mnemonico totale** richiesto da una struttura sintattica = somma dei carichi mnemonici richiesti dalle singole parole **necessarie per completare tale struttura**
    1. DP richiedono VP (in SVO) DP DP DP VP VP VP... è più complessa di DP VP
    2. l'introduzione di un **pronome** che si riferisce ad un'entità già **introdotta** nel discorso è meno complessa dell'introduzione di un **nuovo referente** che dovrà in qualche modo essere integrato nella struttura non ancora completata.

## Grammatiche e Parsing

- ⊙ le grammatiche formali sono (in genere) **dispositivi dichiarativi** che non specificano come un preciso input debba essere analizzato e come una struttura ad albero possa essere costruita.
- ⊙ Fenomeni come il **non determinismo** (la possibilità di scegliere tra più alternative strutturali equivalenti dal punto di vista dell'informazione posseduta al momento della scelta) pongono seri problemi da questo punto di vista, poiché potrebbe facilmente accadere che la scelta strutturale fatta possa rivelarsi sbagliata proseguendo nel parsing. Si richiedono quindi strategie chiare per gestire efficientemente queste situazioni.

## Spazio del problema e strategie di ricerca

- ⊙ data una **frase** ed una **grammatica** il compito del parser è dire se la frase può essere riconosciuta dalla grammatica (**URP, Universal Recognition Problem**) e, in caso affermativo, assegnare alla frase un adeguato Indicatore Sintagmatico
- ⊙ Lo **spazio del problema** è l'insieme di tutti gli alberi e sottoalberi possibili che possono essere generati applicando adeguatamente le regole grammaticali

## Spazio del problema e strategie di ricerca

- ⊙ frase:  
*la vecchia legge la regola*
- ⊙ grammatica:

→ (non terminali)	→ (terminali)
S → DP VP	pro → la
VP → V DP	D → la
VP → pro V	AGG → vecchia
DP → D NP	N → vecchia
NP → (AGG) N;	N → legge
	N → regola
	V → legge
	V → regola

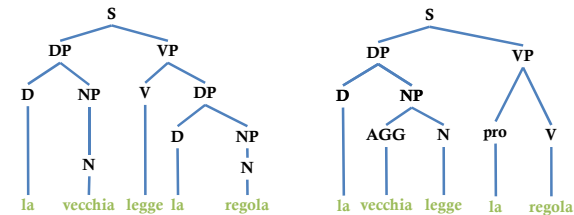


## Spazio del problema e strategie di ricerca

- Due vincoli fondamentali:
  - le regole grammaticali che predicano come da un **nodo radice S** ci siano solo alcune vie di scomposizione possibili per ottenere i nodi terminali;
  - le parole della frase, che ricordano come la (s)composizione di S debba **terminare**
- nel primo caso, se si decide di partire dal **nodo radice S** per generare la struttura adatta alla frase data, si parla di strategia di ricerca **Top-Down o goal-driven**
- nel secondo caso, partendo quindi dalle singole parole e cercando di combinarle in strutture compatibili per giungere ad S, la strategia di ricerca si dice **Bottom-Up, o data-driven**.

## Algoritmo di parsing Top-Down

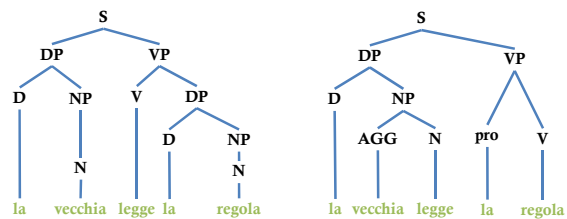
- un semplice algoritmo **top-down** inizia esplorando tutte le possibilità di riscrittura del nodo S che la grammatica offre (assumiamo che ogni albero generato possa poi essere esplorato parallelamente).



- “la regola regola la regola”, “la legge legge la vecchia legge”... saranno scartate solo all’ultimo livello, in fase di mappatura con l’input realmente fornito.

## Algoritmo di parsing Bottom-Up

- primo vero algoritmo associato storicamente all’operazione del parsing (Yngve 55) e probabilmente il più usato (ad esempio nei parsers per i linguaggi di programmazione). La logica che guida la ricerca è quella di partire dagli elementi della frase, cioè dai nodi terminali, e costruire da qui, applicando tutte le regole possibili, una struttura che termini con S:



## Quale è la strategia migliore?

- La strategia **Top-Down** non perde tempo cercando di costruire alberi non consistenti con la grammatica, ma genererà **tutte le alternative possibili prescindendo dall’input**.
- La strategia **Bottom-Up**, sarà **consistente** (almeno localmente) **con l’input** fornito, ma potrà generare per molti livelli **sotto-alberi** che alla fine si riveleranno **non combinabili per ottenere S**.
- Anche se talvolta si possono ottenere risultati comparabili, nella stragrande maggioranza dei casi si deve tener conto di due fondamentali caratteristiche dello spazio del problema:
  - si parte sempre dalla parte dell’albero in cui si dispone dell’**informazione più precisa**
  - si risale l’albero nella direzione in cui il **fattore di ramificazione è minore**.

## Il costo del parallelismo

- Finora abbiamo irrealisticamente assunto che le varie regole potessero essere applicate in **parallelo** (cioè che ogni ramificazione dello spazio del problema potesse essere espansa in contemporanea). In realtà la quantità di memoria richiesta per registrare gli stati dello spazio di un problema creato da un parser che usa una grammatica credibile sarebbe enorme.
- Di solito la soluzione “**brute-force**” avviene attraverso l’esplorazione per **ampiezza (breadth-first)** o per **profondità (deep-first)**: con quest’ultimo metodo si esplora lo spazio del problema **incrementalmente**, espandendo fino ai nodi terminali, un nodo alla volta, solitamente **da sinistra a destra (top-down, depth-first, left-to-right parser)**. Ogni volta che l’operazione fallisce, si sceglie un nuovo percorso iniziando dalla ramificazione più recente.
- Questo tipo di ricerche sono comunque **cieche**, nel senso che **non esistono euristiche** che suggeriscono, di fronte ad un’alternativa, la scelta migliore e soprattutto non si rendono conto dell’errore finché l’espansione non è stata completamente effettuata.

## Algoritmo di parsing LEFT CORNER

- idea di base**  
combinare una strategia Top-Down per la generazione di strutture filtrando le soluzioni con considerazioni di natura Bottom-Up.
- regola dell’angolo sinistro (left-corner)**  
ogni categoria alla fine verrà riscritta come una serie di parole linearmente ordinate e conoscere la prima parola della serie aiuterà a prevedere la categoria o almeno ad escludere soluzioni inconsistenti con questa parola  
Formalmente si può dire che B è l’angolo sinistro della categoria A  
se  $A \rightarrow * B \rightarrow \alpha$ .
- Da tale strategia si ottengono i risultati migliori se preliminarmente viene compilata una tabella che direttamente associa alle categoria un loro possibile **left-corner**:

categoria	S	DP	VP
<i>left-corner</i>	D, N <sub>proprio</sub> , V	D	aux, V

## Alcuni problemi irrisolti

- ricorsività a sinistra**  
una grammatica si dice ricorsiva a sinistra se ammette una regola del tipo:  
 $A \rightarrow * A\alpha$  (es. DP  $\rightarrow$  DP PP)
- ambiguità**  
a vari livelli si possono trovare molteplici strutture che soddisfano i requisiti dell’input e delle regole grammaticali:
  - attaccamento dei PP** (ho visto l’uomo con il cannocchiale)
  - coordinazione** (papaveri e paperi rossi)
 È stato calcolato (Church e Patil 82) che il numero di strutture possibili in corrispondenza di PP cresce **esponenzialmente** con il numero di PP introdotti (se con 3 PP si hanno fino a 5 NP possibili, con 6 PP si arriva a 469 NP possibili... con 8 a 4867).

## Alcuni problemi irrisolti

### b. ambiguità (...continua)

Da una parte il Top-Down parser, se perseguisse il suo compito efficientemente, si dovrebbe **fermare appena recupera una struttura possibile** (nel caso di 3 PP ha una possibilità su 5 di individuare la soluzione più probabile), dall’altra sarebbe **imprudente lasciarlo considerare tutte le strutture generabili**.

L’ambiguità (quella lessicale soprattutto: “vecchia” è Aggettivo o Nome?) causa facilmente errori. La soluzione **deep-first, left-to-right** potrebbe rivelarsi in tal senso **estremamente inefficiente in caso di ambiguità iniziali**.

### c. inefficienza nel ripetere l’analisi dei sottoalberi

il backtracking causato da un errore di parsing, può facilmente provocare il disfacimento di una porzione di albero che in realtà sarebbe costruita bene e che verrà ricostruita tale e quale al prossimo tentativo, come nel caso seguente:

*un volo da Roma per Milano su un 747*

## Programmazione Dinamica

- ⊙ Per **programmazione dinamica** (**dynamic programming**) si intende un approccio che fa uso sistematico di tavole per la registrazione di soluzioni ai sottoproblemi incontrati.
- ⊙ Una volta risolti tutti i **sottoproblemi** (nel caso del parsing sintattico, tutti i sottoalberi), la soluzione al problema globale consiste nel **combinare adeguatamente le singole soluzioni trovate**. Tale metodo risulta essere molto più efficiente degli approcci precedenti e risolve, in linea di principio, almeno il problema dell'inefficienza del ripetere l'analisi corretta dei sottoalberi.

## Programmazione Dinamica: l'algoritmo di Earley

- ⊙ L'**algoritmo di Earley** (Earley 1970) è un esempio classico di programmazione dinamica che usa un approccio top-down parallelo.
- ⊙ La complessità del problema, in linea teorica sempre **NP-hard**, viene **ridotta a polinomiale** (nel peggiore dei casi abbiamo  $O(n^3)$ ) eliminando le soluzioni ripetitive dei sottoproblemi dovuti al backtracking da una struttura scorretta.
- ⊙ L'algoritmo è caratterizzato da un **unico esame, da sinistra a destra**, dell'input che permette di riempire una struttura dati chiamata **grafo** (**chart**) che avrà  $n+1$  entrate, con  $n$  uguale al numero delle parole in input.
- ⊙ Le entrate corrispondono alle **posizioni tra le parole** dell'input (comprese la posizione iniziale, prima della prima parola, e quella finale dopo l'ultima). Per ogni posizione il chart conterrà una lista esaustiva delle strutture fin lì generate ed utilizzabili per ulteriori elaborazioni che cercheranno di integrare posizioni successive. La rappresentazione compatta di questa informazione permette di sfruttare efficientemente, **senza doverla ricomputare in caso di backtrack**, ogni struttura ben formata associata ad un input parziale.

## Programmazione Dinamica: l'algoritmo di Earley

- ⊙ In concreto quindi ogni entrata del **chart** avrà tre tipi di informazioni:
  - un **sottoalbero** corrispondente ad una **singola regola grammaticale**
  - l'**informazione del progresso fatto** al fine di completare il sottoalbero in questione (solitamente si usa un punto • nella parte destra della regola per indicare la posizione a cui si è giunti, la "**regola puntata**" si dice quindi **dotted rule**)
  - la **posizione** del sottoalbero **in relazione all'input** (definito da due numeri indicanti la posizione di inizio della regola e quella dove si trova il punto; es. DP  $\rightarrow$  D • NP [0,1])

## Programmazione Dinamica: l'algoritmo di Earley

- ⊙ L'algoritmo procede con tre operazioni fondamentali:
  - **Previsione (Predictor)**  
crea nuovi stati nell'entrata corrente del chart, rappresentando le aspettative top-down della grammatica; verrà quindi creato un numero di stati uguale alle possibilità di espansione di ogni nodo non terminale nella grammatica.  
es. S  $\rightarrow$  • DP VP [0,0]      DP  $\rightarrow$  • D NP [0,0]
  - **Scansione (Scanner)**  
verifica se nell'input esiste, nella posizione adeguata, una parola la cui categoria combacia con quella prevista dallo stato a cui la regola si trova. Se il confronto è positivo, la scansione produce un nuovo stato in cui l'indice di posizione viene spostato dopo la parola riconosciuta. Tale stato verrà aggiunto all'entrata successiva del chart.  
es. DP  $\rightarrow$  • D NP [0,0] (scansione articolo) se D  $\rightarrow$  articolo: DP  $\rightarrow$  D • NP [0,1] (entrata successiva del chart)

## Programmazione Dinamica: l'algoritmo di Earley

⊙ L'algoritmo procede con tre operazioni fondamentali:

- **Completamento (Completer)**

quando l'indicatore di posizione raggiunge l'estrema destra della regola questa procedura riconosce che un sintagma significativo è stato riconosciuto e verifica se questo avvenuto riconoscimento è utile per completare qualche altra regola rimasta in attesa di quella categoria: ad esempio se nella situazione precedente viene completata positivamente la regola NP → AGG N • [1,3], l'operazione di completamento cercherà tutti gli stati che terminavano nella posizione 1 e che aspettavano un NP per completare la regola. Ma questo era proprio il caso di DP → D • NP [0,1]; quindi il riconoscimento del NP aggiunge anche (all'entrata corrente) lo stato DP → D NP • [0,3];

## Programmazione Dinamica: l'algoritmo di Earley

```

funzione Earley-Parser( stringa_parole, grammatica ) restituisce grafo
  metti_in_coda( ( γ → •S, [0,0] ), grafo[0] )
  per ogni i tale che ( i va da 0 a lunghezza( stringa_parole ) ) esegui
    per ogni stato tale che ( stato è in grafo[i] ) esegui
      se ( incompleto(stato) e non parte_della_frase
        ( prossima_categoria(stato) ) )
        allora esegui Previsione ( stato )
      altrimenti se ( incompleto(stato) e
        parte_della_frase( prossima_categoria(stato) ) )
        allora esegui Scansione ( stato )
      altrimenti esegui Completamento ( stato )
    fine
  fine
restituisce grafo
  
```

## Programmazione Dinamica: l'algoritmo di Earley

```

procedura Previsione ( A → α•Bβ, [i,j] )
  per ogni ( B → γ ) tale che ( Regole_della_Grammatica( B, grammatica ) ) esegui
    metti_in_coda( B → •γ, [i,j], grafo[j] )
  fine
procedura Scansione ( A → α•Bβ, [i,j] )
  se ( B è parte_della_frase( parola[j] ) ) allora esegui
    metti_in_coda( B → parola[j], [j,j+1], grafo[j+1] )
  fine
procedura Completamento ( B → γ•, [j,k] )
  per ogni ( A → α•Bβ, [i,j] ) tale che ( ( A → α•Bβ, [i,j] ) è nel grafo[i] ) esegui
    metti_in_coda( A → αB•β, [j,k], grafo[k] )
  fine
procedura metti_in_coda ( stato, entrata_grafo )
  se ( stato non è già nell'entrata_grafo ) allora esegui
    inserisci( stato, entrata_grafo )
  fine
  
```

## Alcune considerazioni su efficienza e prestazioni

- ⊙ Mentre una grammatica può prescindere da **limiti spazio/temporali** e concentrarsi solo sull'appropriatezza del modello descrittivo fornito, l'architettura del **parser** deve invece crucialmente tener conto di questi vincoli. Questa è una delle principali ragioni per cui **ad una determinata grammatica non corrisponde uno ed un solo parser**. La scelta del "più adatto" spesso richiede valutazioni differenti
- ⊙ Si può parlare di **token trasparenza** (Miller e Chomsky 63) o di **isomorfismo stretto** (ipotesi nulla) quando il parser rispecchia esattamente i passi derivazionali ipotizzati nella grammatica. (Slobin, 1966, mostra in realtà che, contrariamente a quanto si potrebbe pensare, certe costruzioni passive richiedono meno tempo per essere processate rispetto a quelle attive)
- ⊙ Si parla invece di **type transparency** (Bresnan 78) quando alle varie proprietà grammaticali sono associate differenti regole nel modello computazionale che globalmente riproducono la struttura dell'ipotetico parser umano.

## Alcune considerazioni su efficienza e prestazioni

- ⊙ Berwick e Weinberg (83, 84) introducono il concetto di **covering grammars**: un modello della grammatica proposto per il parsing può comunque essere una realizzazione della competenza linguistica purché questo riesca a modellare lo stesso linguaggio per cui la competenza linguistica umana è teoricamente adeguata. Questo modello non sarà psicologicamente plausibile, ma potrà essere ottimale in senso computazionale.

III puntata

## DERIVAZIONI MINIMALI(STE), MEMORIA E INTERVENIENZA

## Grammatiche Minimaliste

- ⊙ Stabler's (1997) formalization of a **Minimalist Grammar, MG** (Chomsky 1995) as a 4-tuple  $(V, Cat, Lex, F)$  such that:

$V$  is a finite set of non-syntactic features,  $(P \cup I)$  where

$P$  are phonetic features and  $I$  are semantic ones;

$Cat$  is a finite set of syntactic features,

$Cat = (base \cup select \cup licensors \cup licensees)$  where

**base** are standard categories {comp, tense, verb, noun ...},

**select** specify a selection requirement  $\{=x \mid x \text{ base}\}$

**licensees** force phrasal movement  $\{-wh, -case \dots\}$ ,

**licensors** satisfy licensee requirements  $\{+wh, +case \dots\}$

$Lex$  is a finite set of expressions built from  $V$  and  $Cat$  (the lexicon);

$F$  is a set of two partial functions from tuples of expressions to expressions :  $\{merge, move\}$ ;

## Grammatiche Minimaliste

$V = P = \{/what/, /did/, /you/, /see/\},$   
 $I = \{[what], [did], [you], [see]\}$

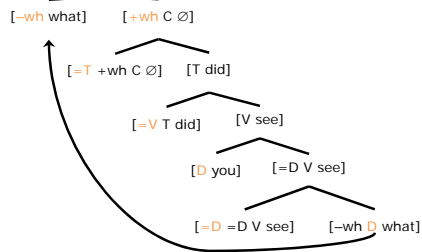
$Cat = base = \{D, N, V, T, C\}$   
 $select = \{=D, =N, =V, =T, =C\}$   
 $licensors = \{+wh\}$   
 $licensees = \{-wh\}$

$Lex = \{[-wh D \text{ what}], [=V T \text{ did}], [D \text{ you}], [=D =D V \text{ see}],$   
 $[=T +wh C \emptyset]\}$

$F = \{merge, move\}$  such that:  
 $merge ([=F X], [F Y]) = [{}_x X Y]$   
 ("simple merge" on the right, "complex merge" on the left)  
 $move ([+g X], [W [-g Y]]) = [[{}_x Y X] W, t_y]$

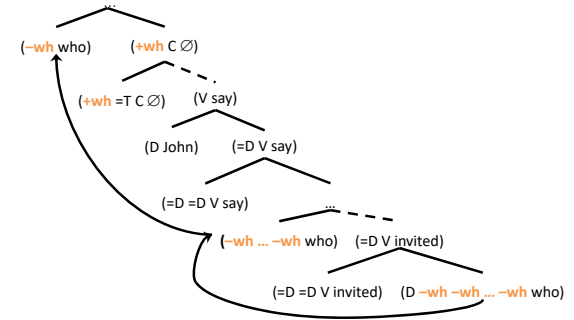
## Grammatical Minimalism

- merge ( $[=D =D V \text{ see}]$ ,  $[-\text{wh } D \text{ what}]$ )  $\rightarrow$   $[_{\text{see}} =D V \text{ see}, -\text{wh what}]$
- merge ( $[D \text{ you}]$ ,  $[=D V \text{ see}, -\text{wh what}]$ )  $\rightarrow$   $[_{\text{see}} \text{ you}, [_{\text{see}} V \text{ see}, -\text{wh what}]]$
- merge ( $[-V T \text{ did}]$ ,  $[_{\text{see}} \text{ you}, [_{\text{see}} V \text{ see}, -\text{wh what}]]$ )  $\rightarrow$   $([_{\text{did}} T \text{ did}, [_{\text{see}} \text{ you}, [_{\text{see}} \text{ see}, -\text{wh what}]]])$
- merge ( $[=T +\text{wh } C \emptyset]$ ,  $([_{\text{did}} T \text{ did}, [_{\text{see}} \text{ you}, [_{\text{see}} \text{ see}, -\text{wh what}]]])$ )  $\rightarrow$   $([_{\text{C}} +\text{wh } C \emptyset, ([_{\text{did}} T \text{ did}, [_{\text{see}} \text{ you}, [_{\text{see}} \text{ see}, -\text{wh what}]])])$
- move ( $([_{\text{C}} +\text{wh } C \emptyset, ([_{\text{did}} T \text{ did}, [_{\text{see}} \text{ you}, [_{\text{see}} \text{ see}, -\text{wh what}]])])$ )  $\rightarrow$   $([_{\text{C}} \emptyset, ([_{\text{C}} \text{ What } C \emptyset, ([_{\text{did}} T \text{ did}, [_{\text{see}} \text{ you}, [_{\text{see}} \text{ see}, t_{\text{what}}]])])$ )



## MG: problems with successive cyclicity

- Wh- successive cyclic movement

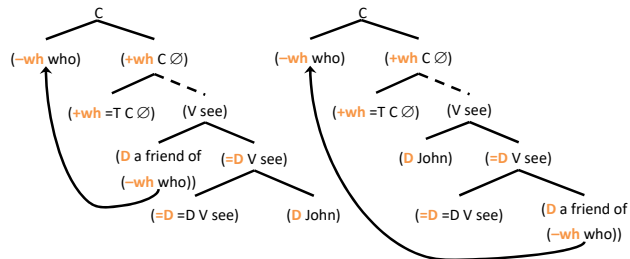


Phase-based Minimalist Grammars (PMGs)

C. Chesi

## MG: how explaining islandhood?

- No difference in picking up an element from a **subject** or an **object** (idem for **RCs** and **Adjuncts**)



Phase-based Minimalist Grammars (PMGs)

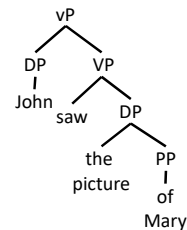
C. Chesi

## Representations vs. Derivations

- “the computational system takes representations of a given format and modifies them” (Chomsky 1993:6)
- The order of Structure Building Operation is abstract with “no temporal interpretation implied” (Chomsky 1995:380)
- Derivation by Phase** (Chomsky 2005-08): a phase is a Syntactic Object built assuming Structure Building Operations (**Merge** and **Move**) over a finite set of Lexical Item (Lexical Array, aka **Numeration**) CP and vP are phases (maybe DP)

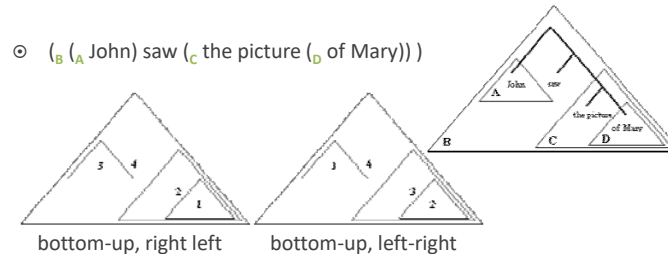
## Derivations: some logical possibilities

- ⊙ ((John) saw ((the picture) (of Mary)))



## Derivations: Local Relations

- ⊙ (B (A John) saw (C the picture (D of Mary)))



Phase-based Minimalist Grammars (PMGs)

C. Chesi

## Processing Object Relatives (ORs)

- ⊙ Bever (1970)  
double embedding is not always nearly impossible to process (Miller & Chomsky 1963):
  - The reporter the politician the commentator** met trusts said the president won't resign.
  - The reporter everyone I** met trusts said the president won't resign.

Intro to linguistic computation

C. Chesi

## Processing Object Relatives (ORs)

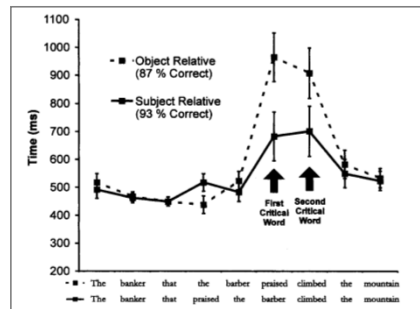
- ⊙ Gordon, Hendrick & Johnson (2001)  
**working memory** request is evaluated by studying **reading time (RT)** and **comprehension accuracy** in self-paced reading experiments comparing critical regions of various kinds of **Relative Clauses**:
  - ⊙ **Experiment 1** (materials): **SRs** (a) and **ORs** (b)
    - The banker** [that \_ praised **the barber** ] climbed the mountain
    - The banker** [that **the barber** praised \_ ] climbed the mountain

Intro to linguistic computation

C. Chesi

## Processing Object Relatives (ORs)

- Gordon et al. (2001) - **Experiment 1** (results)



Intro to linguistic computation

C. Chesi

## Processing Object Relatives (ORs)

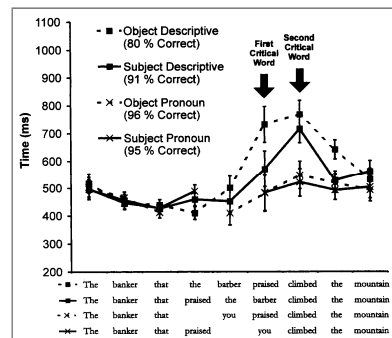
- Gordon et al. (2001) - **Experiment 2**  
complexity can be mitigated by varying the RC **Subject typology** (reading time (RT) and **comprehension accuracy** in self-paced reading experiments are tested, as before):
- **Experiment 2** (materials): **DP** (a) vs. **Pro** (b)
  - The banker** [that **the barber** praised \_ ] climbed the mountain
  - The banker** [that **you** praised \_ ] climbed the mountain

Intro to linguistic computation

C. Chesi

## Processing Object Relatives (ORs)

- Gordon et al. (2001) - **Experiment 2** (results)



Intro to linguistic computation

C. Chesi

## Processing Object Relatives (ORs)

- Gordon et al. (2001) - **Experiment 3** (materials):  
**DP** (a) vs. **proper names** (b)
  - The banker** [that **the barber** praised \_ ] climbed the mountain
  - The banker** [that **Ben** praised \_ ] climbed the mountain

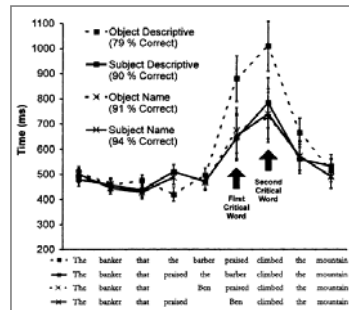
Intro to linguistic computation

C. Chesi



## Processing Object Relatives (ORs)

- Gordon et al. (2001) - **Experiment 3** (results)



Intro to linguistic computation

C. Chesi

## Processing Object Clefts

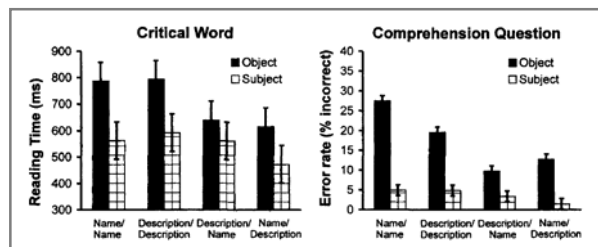
- Gordon et al. (2001) - **Experiment 4** (materials):  
Subject vs. Object Clefts X **DP** vs. **proper names**
  - It was **the banker** that **the lawyer** saw \_ in the parking lot
  - It was **the banker** that **Bill** saw \_ in the parking lot
  - It was **John** that **the lawyer** saw \_ in the parking lot
  - It was **John** that **Bill** saw \_ in the parking lot

Intro to linguistic computation

C. Chesi

## Processing Object Clefts

- Gordon et al. (2001) - **Experiment 4** (results):



Intro to linguistic computation

C. Chesi

## Explaining complexity

- **Role-determinant** accounts (MacWhinney & Pleh 1988)
  - **Double role** for the **RC head**: **subject** in the matrix sentence, **object** in the RC:  
**The banker** [that the barber **praised** \_ ] **climbed** the mountain (OR)
- **Memory-load** accounts (Ford 1983, MacWhinney 1987, Wanner & Maratsos 1978)
  - **The RC head** must be **kept in memory longer** in OR before being integrated:  
**The banker** [that **praised** the barber] climbed ... (SR)  
**The banker** [that the barber **praised** \_ ] climbed ... (OR)

Intro to linguistic computation

C. Chesi

## Explaining complexity

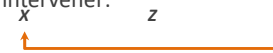
- ⊙ **Linguistic Integration Cost** (Gibson 1998:12-13)
  - Processing difficulty is proportional to the distance expressed in terms of number of **intervening discourse referents**, following a “referentiality hierarchy”: **descriptions > (short) names > referential pronouns > indexical pronouns**
- ⊙ **Similarity based accounts** (Gordon et al. 2001)
  - Having **two DPs of the same kind** stored in **memory** makes the OR more complex than SR. This models **memory interference** during encoding, storage and retrieval (Crowder 1976)

Intro to linguistic computation

C. Chesi

## Explaining complexity

- ⊙ More on **Similarity based accounts** (Gordon et al. 2001)
  - It might be able to explain why SR vs. OR asymmetry disappears with RC subject pro/proper names (those DPs are legal heads only for clefts)
- ⊙ **Intervention effects** (Grillo 2008, Friedmann et al. 2009, Rizzi 1990)
  - Processing difficulty is proportional to the number and kind of relevant features shared between the moved item and any possible intervener:



Intro to linguistic computation

C. Chesi

## Explaining complexity

- ⊙ More on **Intervention effects** (Friedmann et al. 2009)
  - **Identity** (**bad** for adults, **bad** for children)
  - **Inclusion** (**ok** for adults, **bad** for children)
  - **Disjunction** (**ok** for adults, **ok** for children)

Intro to linguistic computation

C. Chesi

## Comparing Object Clefts

- ⊙ Warren & Gibson (2005) - **Experiment** (materials): **definite descriptions** vs. **proper names** vs. **pronouns**
  - It was **the banker** that **the lawyer** avoided \_ at the party
  - It was **the banker** that **Dan** avoided \_ at the party
  - It was **the banker** that **we** avoided \_ at the party
  - It was **Patricia** that **the lawyer** avoided \_ at the party
  - It was **Patricia** that **Dan** avoided \_ at the party
  - It was **Patricia** that **we** avoided \_ at the party
  - It was **you** that **the lawyer** avoided \_ at the party
  - It was **you** that **Dan** avoided \_ at the party
  - It was **you** that **we** avoided \_ at the party

Intro to linguistic computation

C. Chesi

## Comparing Object Clefts

- Warren & Gibson (2005) - **results** (Tessa Warren P.C.)

**D** = definite description (e.g. **the banker**)

**N** = proper names (e.g. **Dan**)

**P** = pronouns (e.g. **you**)

condition	D-D	D-N	D-P	N-D	N-N	N-P	P-D	P-N	P-P
Read. time (SE) ms	365 (19)	319 (12)	306 (14)	348 (18)	347 (21)	291 (14)	348 (18)	311 (15)	291 (13)

Intro to linguistic computation

C. Chesi

## On the superiority of intervention-based accounts

- Assuming that **Definite Description** = {**+NP, N**}, **Proper Names** = {**+NP, N<sub>Proper</sub>**}, **pro** = {} (Belletti & Rizzi 2013), **Intervention effects** are predicted to be stronger in matching **D-D** and **N-N** condition (against memory-load accounts), while **P-P** is expected not to be critical (because of the **+NP** absence):

condition	D-D	D-N	D-P	N-D	N-N	N-P	P-D	P-N	P-P
Read. time (SE) ms	365 (19)	319 (12)	306 (14)	348 (18)	347 (21)	291 (14)	348 (18)	311 (15)	291 (13)
prediction	hard	?	easy	?	hard	easy	easy	easy	easy

Intro to linguistic computation

C. Chesi

## Some problems with the intervention-based account

- Features triggering movement are those relevant for intervention (Friedmann et al. 2009:82), but:
  - +R** feature causing Object movement in ORs (or **+Foc** in OCs) is not present on Subject;
  - Neither the **“lexical restriction”** nor **phi-features** trigger any movement in **ORs** or **OCs**
  - The **“lexical restriction”** should be not accessible at the **edge of the DP**, where features triggering movement should be located (but see Belletti & Rizzi 2013, next slide)
  - Why slow-down is observed at **verb segment**?

Intro to linguistic computation

C. Chesi

## Some problems with the intervention-based account

- Belletti & Rizzi 2013:
  - Evidence that lexically restricted wh-items occupy different positions in the left periphery (Munaro 1999):
    - Con **che tosàt** à-tu parlà?  
*with which boy did you speak?*
    - Avé-o parlà de **chi**?  
*Have you spoken of whom?*

Intro to linguistic computation

C. Chesi

## Feature Retrieval Cost (FRC) Why do we need it? (a summary)

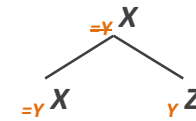
- ⊙ An “integration cost” (cf. Gibson 1998) is not enough
  - È **il bambino** che **il signore** ha salutato ...
  - È **Luigi** che **Gianni** ha salutato ...
- ⊙ **Intervention-based** accounts are not “gradable” (no quantitative, precise, measurements)
- ⊙ **Bottom-Up** standard theories do not make clear **predictions on processing**: they predict **what** creates complexity, but not **when, why** and **how exactly** in **parsing** and **generation**?

Intro to linguistic computation

C. Chesì

## Processing-friendly Phase-based Minimalist Grammar

- ⊙ Common restriction on **Merge**:
  - Given two lexical items  $[-_Y X]$  and  $[_Y Z]$  such that  $X$  selects  $Z$ , then:



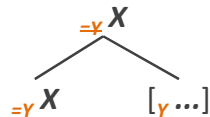
- $[-_Y X]$  is processed before  $Y$
- When  $[-_Y X]$  is processed, an **expectation** for  $[_Y \dots]$  is created

Intro to linguistic computation

C. Chesì

## Processing-friendly Phase-based Minimalist Grammar

- ⊙ A **Phase** is the minimal computational domains within which a selection requirement must be satisfied:
  - Given a lexical item  $[-_Y X]$ ,  $[_Y \dots]$  is the **selected phase**:



- **Merge** reduces to **lexical selection** (or **unification**) (e.g.  $[_Y Z]$  insertion)

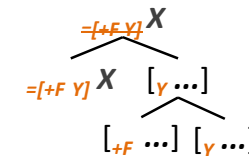
Intro to linguistic computation

C. Chesì

## Processing-friendly Phase-based Minimalist Grammar

- ⊙ If we assume that selection can include both functional features  $(+F)$  and lexical features  $(Y)$  at the same time, a **Phase** becomes a subtree to be expanded:

- Given a lexical item  $[-_{+F Y} X]$ ,  $[-_{+F Y} \dots]$  is the **selected phase**:



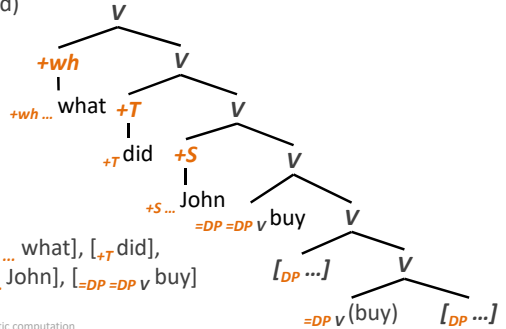
- $[-_{+F Y} \dots]$  is an **extended projection** of a **lexical category Y** (e.g. a **DP** is an extended projection of **N**, i.e.  $[-_{+D N}]$ )

Intro to linguistic computation

C. Chesì

## Processing-friendly Phase-based Minimalist Grammar

- Both a declarative sentence [+S +T V] and a *wh*-question [+wh +T +S V] are phases (i.e. extended projections of a V head)



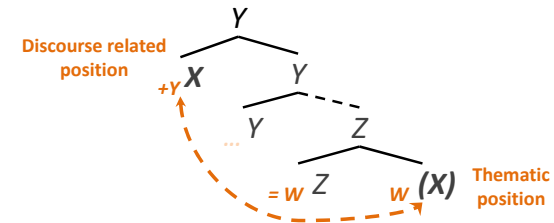
- [+wh ... what], [+T did], [+S ... John], [=DP =DP<sub>V</sub> buy]

Intro to linguistic computation

C. Chesi

## Processing-friendly Phase-based Minimalist Grammar

- Common trigger for **Move**:
  - An item [+Y ... W X], in a given structure, must be moved if it can not be **fully interpreted** in its insertion position:

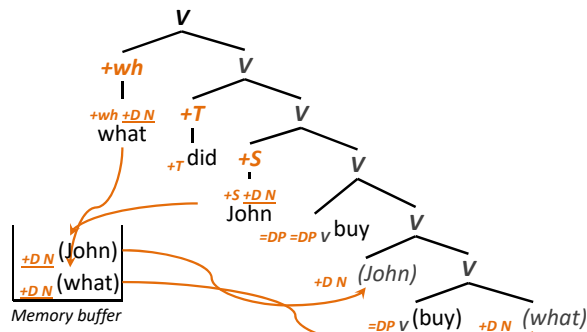


Intro to linguistic computation

C. Chesi

## Processing-friendly Phase-based Minimalist Grammar

- [+wh +D<sub>N</sub> what], [+T did], [+S +D<sub>N</sub> John], [=DP =DP<sub>V</sub> buy]



Intro to linguistic computation

C. Chesi

## Processing-friendly Phase-based Minimalist Grammar

- The derivation unfolds **Top-Down** and **Left-Right**
- Unexpected features** trigger movement
- Phases** restrict the domain in which a non-local dependency must be satisfied
- Last-In-First-Out memory buffer**, as a first approximation, is used to store and retrieve items for non-local dependencies (memory buffer must be **empty** at the end of the derivation)
- The order in which phases are expanded makes a difference: the last **selected phase** has a special status (**sequential phase**) while phases that are not the last selected ones (e.g. phases that results from expansion of functional features) qualifies as **nested phases** (Bianchi & Chesi 2006)

Intro to linguistic computation

C. Chesi

## Deriving OCs Top-Down

- In Object Clefts (OCs), the copula selects a truncated CP (Belletti 2008):

... BE [<sub>CP</sub> Force [<sub>FocP</sub> ... [<sub>FinP</sub> che [<sub>TP</sub> Subject ... **Object** ] ] ] ]

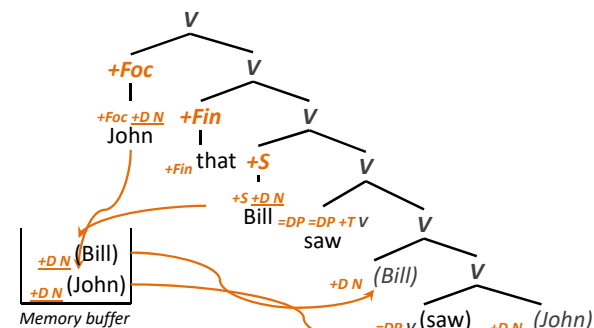
- Reduced CP (CP<sub>r</sub>) = [+Foc +Fin +S +T V]

Intro to linguistic computation

C. Chesì

## Deriving OCs Top-Down

- It [<sub>CP</sub> ... was] [<sub>CP<sub>r</sub></sub> John that Bill saw]



Intro to linguistic computation

C. Chesì

## Cue-based retrieval and intervention

- **interference** is the major constraint on accessing information in memory (Anderson & Neely 1996; Crowder 1976; see Nairne 2002 for a review).
- the locus of the interference effect is at **retrieval**, with **little or no effect** on memory **encoding** or **storage** (Dillon & Bittner 1975; Gardiner et al. 1972; Tehan & Humphreys 1996)
- **Content-addressable memory** (e.g. **memory load paradigm**, Van Dyke & McElree 2006), no **exhaustive search**, no **delay**
- **Search of Associative Memory (SAM)** model (Gillund & Shiffrin 1984)

$$P(I_j | Q_1, \dots, Q_n) = \frac{\prod_{j=1}^m S(Q_j, I_j)^{w_j}}{\sum_{k=1}^N \prod_{j=1}^m S(Q_j, I_k)^{w_j}}$$

Intro to linguistic computation

C. Chesì

## On DP features (and structure)

- Elbourne (2005)  
[[**THE** *t*] NP]
- Zamparelli (1995-2000)  
[<sub>SDP</sub> *Strong QP* [<sub>PDP</sub> *Weak QP* [<sub>KIP</sub> (*Restrictive Adj*) [<sub>NP</sub> *Noun*]]]]
- Longobardi (1994-2005), a rough summary:
  - **Definite Descriptions** [<sub>D</sub> the [<sub>N</sub> man]]
  - **Proper Names** [<sub>D</sub> John<sub>i</sub> [<sub>N</sub> *t*<sub>i</sub>]]
  - **Pronouns** [<sub>D</sub> you [<sub>N</sub> ∅ ]]

Intro to linguistic computation

C. Chesì

## Relevant DP features Definite Descriptions & Proper Names

- Both **proper names** and **common nouns** have category **N**
  - N in situ* (common nouns)**     ***N-to-D raising***
  - Il mio Gianni (Il mio amico)     \*mio Gianni
  - La sola Maria (la sola amica)     Maria sola (\*l'amica sola)
- Two different kinds of N: **N<sub>proper</sub>**, **N<sub>(common)</sub>**

Intro to linguistic computation

C. Chesì

## Relevant DP features On D and Pronouns

- Both **determiners** and **personal pronouns** introduce a “referential pointer” to an individual constant or variable in the domain of discourse
- Pro are NP-ellipsis licensors (they can be used as determiners «we italians»): [<sub>D</sub> noi [<sub>N</sub> italiani]]  
(**D** introduces an *index*, that bounds a variable predicated in N)
- (More) features on **pro**:
  - 1<sup>st</sup>** and **2<sup>nd</sup>** person (highly accessible referents) vs. **3<sup>rd</sup>** person (**default person**, context-determined referent)
  - case**

Intro to linguistic computation

C. Chesì

## Relevant DP features

- Definite descriptions:**     **{+D, +num, N}**
- Proper names:**     **{+D, +num, N<sub>prop</sub>}**
- Pronouns:**     **{+D, +case, +pers, +num}**

Intro to linguistic computation

C. Chesì

## Feature Retrieval Cost (FRC) metrics at work

- Cost function (at **X** given **m<sub>x</sub>** items to be retrieved from memory)

$$C_{\text{FRC}}(X) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

- m** = number of items stored in memory at retrieval
- nF** = number of features characterizing the argument to be retrieved that are non-distinct in memory (i.e. also present in other objects in memory)
- dF** = number of distinct cued features (e.g. agreement and case features probed by the verb)

Intro to linguistic computation

C. Chesì

## Feature Retrieval Cost (FRC) metrics at work

$$C_{\text{FRC}}(x) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

⊙ *D-D* matching

it was **the lawyer**<sub>{+D, +num\_sing, N}</sub> who **the businessman**<sub>{+D, +num\_sing, N}</sub> avoided...

$$C_{\text{FRC}}(\text{avoided}) = 16$$

that is **16 · 1**:

**16** for retrieving **the businessman**,

**nF=3, m=2** (because two *Ds* are in memory at that retrieval time), and  
**dF=0** (because no feature is cued by the verb distinguishing one *D*  
from the other);

**1** for retrieving **the lawyer**,  
since **nF=0, m=1** and **dF=0**

## Feature Retrieval Cost (FRC) metrics at work

$$C_{\text{FRC}}(x) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

⊙ *N-N* matching

it was **Dan**<sub>{+D, +num\_sing, N\_prop}</sub> who **Patricia**<sub>{+D, +num\_sing, N\_prop}</sub> avoided...

$$C_{\text{FRC}}(\text{avoided}) = 16$$

that is **16 · 1**:

**16** for retrieving **Dan**,

**nF=3, m=2** (because two *Ds* are in memory at that retrieval time), a  
nd **dF=0** (because no feature is cued by the verb distinguishing one *D*  
from the other);

**1** for retrieving **Patricia**,  
since **nF=0, m=1** and **dF=0**

## Feature Retrieval Cost (FRC) metrics at work

$$C_{\text{FRC}}(x) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

⊙ *P-P* matching

it was **you**<sub>{+D, +pers\_ll, +num\_sing, +case}</sub> who **we**<sub>{+D, +pers\_l, +num\_plur, +case\_nom}</sub> avoided...

$$C_{\text{FRC}}(\text{avoided}) = 2$$

that is **2 · 1**:

**2** for the **we**,

**nF=1, m=2** and **dF=1** (**number, person** and **case** mismatches are  
always present; **case** is cued by the verb),

**1** for retrieving **you**,  
**nF=0, m=1** and **dF=0** for the object pronoun

## Feature Retrieval Cost (FRC) metrics at work

$$C_{\text{FRC}}(x) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

⊙ *D-N* matching

it was **the lawyer**<sub>{+D, +num\_sing, N}</sub> who **Patricia**<sub>{+D, +num\_sing, N\_prop}</sub> avoided...

$$C_{\text{FRC}}(\text{avoided}) = 12,25$$

that is **12,25 · 1**:

**12,25** for the **the lawyer**,

**nF=2.5, m=2, dF=0**

(*N<sub>prop</sub>* vs. *N* counts as half because of  
movement)



## Feature Retrieval Cost (FRC) metrics at work

$$C_{\text{FRC}}(x) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

⊙ *P-D* condition

it was **you**<sub>{+D, +pers\_ll, +num\_sing, +case}</sub> who **the businessman**<sub>{+D, +num\_sing, N}</sub> avoided...

$$C_{\text{FRC}}(\text{avoided}) = 9$$

that is  $9 \cdot 1$ :  
**9** for the **the businessman**,  
 $nF=2, m=2, dF=0$

## Feature Retrieval Cost (FRC) metrics at work

$$C_{\text{FRC}}(x) = \prod_{i=1}^{m_x} \frac{(1+nF_i)^{m_i}}{(1+dF_i)}$$

⊙ *D-P* condition

it was **the lawyer**<sub>{+D, +num\_sing, N}</sub> who **we**<sub>{+D, +pers\_l, +num\_plur, +case\_nom}</sub> avoided...

$$C_{\text{FRC}}(\text{avoided}) = 4,5$$

that is  $4,5 \cdot 1$ :  
**4,5** for the **we**,  
 $nF=2, m=2, dF=1$  (case is cued)

## Feature Retrieval Cost (FRC) metrics at work

The complete prediction set:

condition	D-D	D-N	D-P	N-D	N-N	N-P	P-D	P-N	P-P
Read. time (SE) ms	365 (19)	319 (12)	306 (14)	348 (18)	347 (21)	291 (14)	348 (18)	311 (15)	291 (13)
prediction	16	12,25	4,5	12,25	16	4,5	9	9	1

## Conclusion

- ⊙ I rephrased the intervention-based idea (Friedmann et al. 2009) in Top-Down terms, trying to reconcile the **formal account** of intervention (**what**) with **processing evidence** (**when** and **how**)
- ⊙ What permits to express the exact complexity cost is a **Top-down** (that in the end produce a **left-right**) derivation (this way the model fitting can be directly compared with other complexity metrics, e.g. SPLT, Gibson 1998)
- ⊙ The special role of intervention has been expressed in terms of **interference** at **retrieval** (e.g. Van Dyke & McElree 2006)

## Further development

- ⊙ Feature structures (and actual cues) need to be further refined (other features, e.g. **animacy**, Kidd et al. 2007, and **semantic selection**, Gordon et al. 2004, should be considered)
- ⊙ The counterintuitive idea that **Subject** “is harder” to retrieve than **Object** in ORs should receive experimental support
- ⊙ Is it a purely privative system (+/- F) enough?
- ⊙ Doing away with **LIFO structure** which is computationally **OK**, but psycholinguistically odd (cf. **content-addressable memory**).

Intro to linguistic computation

C. Chesi

## Concetti fondamentali del minicorso

- ⊙ Cos'è una **grammatica formale** e cosa serve per specificarne una
  - Regole di riscrittura e ricorsività
  - Restrizioni sulle regole di riscrittura per creare classi di grammatiche generativamente più o meno potenti (gerarchia di Chomsky)
  - Equivalenza tra grammatiche, macchine a stati finiti e push-down automata
  - dove stanno i linguaggi naturali
- ⊙ Che cos'è una **computazione**
  - Cos'è lo spazio del problema e come lo spazio viene esplorato da un algoritmo
  - Come si calcola la complessità di un problema
  - Cosa vuol dire fare parsing di una frase e alcuni algoritmi
- ⊙ Cos'è una grammatica **Top-Down**
  - Competenza e Performance... riunite
  - Ricostruzione di elementi dalle isole
  - Aspettative e regolamentazione

## Domande possibili

- ⊙ Cos'è una grammatica a struttura sintagmatica e come si leggono le regole di riscrittura (e.g. NP → D N)?
- ⊙ Come mai le grammatiche regolari (RG) sono meno potenti di quelle context-free (CFG)?  $a^n b^n$  cosa significa? È una proprietà che possono esprimere le stringhe riconoscibili da una RG?  $XX$  invece? Mentre  $XX^n$ ?
- ⊙ Cosa sono le macchine a stati finiti (FSA) e i push-down automata (PDA) e che linguaggi possono generare/riconoscere?
- ⊙ Come si fa a determinare se una stringa esprime proprietà catturabili o no da un certa classe di grammatiche?
- ⊙ Su che dimensioni si esprime la complessità di un algoritmo?
- ⊙ Come si possono usare le grammatiche context-free per fare parsing?
- ⊙ Quale è una fondamentale differenza tra RG/CFG e grammatiche minimaliste?
- ⊙ Come si può esprimere la complessità in termini di intervenienza?

Intro to linguistic computation

C. Chesi