Linguistica Computazionale – Lezione 7

Parsing (sintattico) II

Unificazione, Principi & Parametri e Minimalismo

Mercoledì 20 Marzo 2007

Cristiano Chesi, chesi@media.unisi.it

Parsing (sintattico) II

- Indice
  - Grammatiche ad unificazione
    - tratti e gerarchie
    - HPSG
    - fenomeni linguistici da catturare
  - Principle & Parameter Parsing
    - dalle regole ai principi
    - un esempio di P&P parser (PAPPI, Fong 1991)
  - Introduzione alle grammatiche minimaliste

Letture, approfondimenti

- **Bibliografia essenziale**
  - Jurafsky & Martin (2000) *Speech & Language Processing*. Prentice Hall, NJ (Cap. 11)
  - Allegranza, V., Mazzini G. (2000) *Linguistica generativa e grammatiche a unificazione*. Paravia scriptorium. (Cap. 2, 4, 5)
- **Approfondimenti**
  - Allegranza, V., Mazzini G. (2000) *Linguistica generativa e grammatiche a unificazione*. Paravia scriptorium.
  - Fong S. (1991) *Computational Properties of Principle-Based Grammatical Theories*. tesi Ph.D. MIT
  - Pollard C. & Sag I. (1994) *Head-Driven Phrase Structure Grammar*. CSLI
  - Stabler E. (1997) *Derivational minimalism*. in Retoré, ed. Logical Aspects of Computational Linguistics. Springer

Problemi ancora aperti

Grammatiche ad Unificazione

- Limiti evidenti delle grammatiche a struttura sintagmatica (sia **context-free** che **context-sensitive**), difficoltà nel catturare certi fenomeni, se non al costo di una proliferazione esponenziale di regole e categorie che porta velocemente all'**ipergeneralizzazione** o alla **sottogeneralizzazione** (impossibilità di catturare strutture in realtà corrette):

1. **accordo**
2. **sottocategorizzazione**
3. **relazioni a distanza**

Soluzioni proposte

- **Grammatiche basate su restrizioni**
  rappresentazione più efficiente e significativa dell'informazione linguistica
- **formalismi leggermente più potenti delle CFG**, con cui render conto in modo
  - **compatto** (quindi più elegante) ed
  - **efficiente**

  delle restrizioni linguistiche imposte da fenomeni produttivi quali quelli precedentemente elencati
- **gerarchie di tratti**
  proprietà aggiuntive rispetto alle regole di riscrittura

(lezione 2)

Formalismi basati su restrizioni (Contraint-based formalisms)

(inizio …)

- L'introduzione dei tratti richiede un test di "**buona costituzione**" oltre alle normali regole di concatenazione derivanti dai dispositivi di riscrittura
- **Unificazione**
  operazione sui tratti di due elementi che si vogliono combinare
  1. verifica di compatibilità
  2. selezione dei tratti che potranno essere ereditati dal nuovo elemento formatosi in seguito all'unione

(lezione 2)

Formalismi basati su restrizioni (Contraint-based formalisms)

(… continua …)

- **Struttura di tratti** (**FS**, **Feature Structures**)
  è un insieme di coppie del tipo **tratto > valore**
  (es. numero > singolare)
- due tipologie di formalizzazione (equivalenti) delle coppie **tratto > valore**:

Matrice di Attributi e Valori
(AVM, Attribute-Value Matrix)

$$\begin{bmatrix} \text{Num} = \text{Sing} \\ \text{Gen} = \text{Femm} \\ \dots \\ \text{Tratto}_n = \text{Valore}_n \end{bmatrix}$$

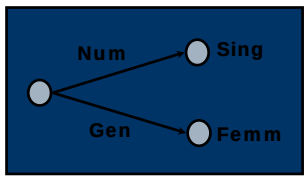


diagramma ad archi orientati ed etichettati
(DAG, Direct Acyclic Graph)

(lezione 2)

Formalismi basati su restrizioni (Contraint-based formalisms)

(… continua …)

Alcune proprietà interessanti delle strutture di tratti:

- **parzialità**, **maggiore** o **minore specificità**, ovvero alcuni elementi posso restare non specificati, ad esempio il genere:

$$N\begin{bmatrix} \text{Num} = \text{Sing} \\ \text{Gen} = [\,] \end{bmatrix}$$

- la struttura delle AVM può essere **rientrante**, cioè un tratto che ha una qualche significatività dal punto di vista empirico, può essere definito da più sottotratti, come nel caso dell'accordo:

$$\begin{bmatrix} \text{Cat} = \text{N} \\ \text{Accordo}\begin{bmatrix} \text{Num} = \text{Sing} \\ \text{Gen} = \text{Femm} \end{bmatrix} \end{bmatrix}$$

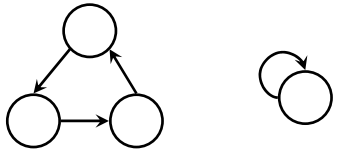
Altre proprietà interessanti delle strutture di tratti:

- **percorsi**, il valore di un tratto è definito in base ad un percorso univoco che lo identifica, cioè una lista di tratti lungo la struttura del tipo: accordo>num>sing
- **condivisione di tipo** (**type sharing**), una struttura può essere condivisa tra più elementi anche se i valori non lo sono
- **condivisione di occorrenza** (**token sharing**), l'occorrenza di un determinato valore può essere condivisa in tal caso può essere indicato con l'uso di un indice, es [1]:

$$\begin{bmatrix} Cat = S \\ testa \begin{bmatrix} Accordo\ [1] \begin{bmatrix} Num = Sing \\ Per = 3 \end{bmatrix} \\ Soggetto \begin{bmatrix} Accordo\ [1] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Alcune proprietà interessanti delle strutture di tratti:

- **significatività empirica**, i tratti sembrano catturare adeguatamente, almeno a livello descrittivo, fenomeni linguisticamente produttivi
- **aciclicità**, i grafi non possono essere ricorsivi



Sussunzione

stabilisce una relazione ordinata tra due strutture di tratti FS; la FS più **generica** sussume quella **specifica**. Si può quindi dire che:
$FS_a \sqsubseteq FS_b$

sse FS_b ha tutti i tratti di FS_a nella stessa configurazione strutturale e con uguali assegnazioni di valore

Unificazione

permette di combinare le informazioni per rappresentarle in formato più compatto e significativo:
$FS_a \sqcup FS_b = FS_x$ (se esiste) tale che FS_x è la più generale delle FS sussunte da FS_a e FS_b

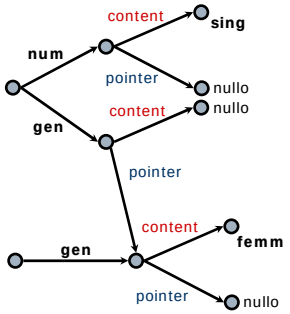
Grammatiche ad unificazione e parsing

- **Algoritmo di unificazione**
  1. input: due strutture di tratti
  2. verifica compatibilità strutture tratti
  3. restituzione di una singola struttura unita
     oppure rifiuto delle strutture in input riconosciute come incompatibili
- **idea di base**: creare un **loop** sui **DAG** da combinare, fino ad esaurimento tratti, creando riferimenti da un DAG all'altro quando strutture e valori sono compatibili.
- piccola **modifica dei DAG**: ad ogni tratto si associa un valore di campo (**content**) e un puntatore (**pointer**); il primo indicherà il valore del tratto nel DAG, il secondo stabilirà un link con un'altra struttura dati compatibile (se esiste, altrimenti sarà nullo).
- **occur check**: per evitare cicli (che occorrerebbero se si cercasse di unire una struttura con un'altra che la contiene già propriamente come sottoparte) si verifica la relazione di dominanza impedendo l'unione nel caso che la struttura che si vuole linkare domini quella da cui si origina il link

Grammatiche ad unificazione e parsing

(… continua …)

Questo il risultato dell'unione dei due DAG modificati **num>sing** ⊔ **gen>femm**:



num > sing
gen > []

gen > femm

Grammatiche ad unificazione e parsing

(… fine)

- Per implementare l'unificazione ad una grammatica CFG si può procedere nel seguente modo:
  1. si **associano strutture di tratti complesse** sia agli elementi lessicali che alle categorie grammaticali
  2. si definiscono composizionalmente delle **regole di combinazione** di questi tratti
  3. si stabiliscono i **vincoli di compatibilità** tra strutture di tratti
- Utilizzando il formalismo introdotto da Shieber (1986) le regole nella grammatica saranno espresse come segue:

  S → **DP VP**
  <DP accordo **= VP** accordo**>**

Grammatiche ad unificazione e parsing - HPSG

(inizio …)

- **HPSG** - Head-Driven Phrase Structure Grammar (Pollard & Sag 1987, 1994)
  - teoria linguistica **integrata** (sintassi + semantica)
  - **rappresentazionale** (come le teorie trasformazionali, es. Government and Binding) non **derivazionale** quindi **non movimento**, ma **structure sharing** (**token identity**)
  - relazioni quali la **dominanza** o il **C-comando** vengono indirettamente espresse dalla **relative obliqueness** (relazione tra testa ed elementi da essa dipendenti)
  - frasi, sintagmi e parole hanno gli attributi [**phon**] (phonology, informazioni fonologiche) e [**synsem**] (syntax-semantics, informazioni sintattiche e semantiche)
  - i sintagmi inoltre posseggono l'attributo [**dtrs**] (daughters, nodi figli)

Grammatiche ad unificazione e parsing - HPSG

(… continua …)

- **tratto** (principio) **testa**
  per quanto riguarda l'**accordo**, si può notare come un nome che possiede un tratto singolare all'interno di un NP costringa l'intero NP ad essere accordato con tale tratto.
  In genere è sempre la **testa** del sintagma lessicale che determina i valori di accordo. Per catturare questa generalizzazione si deve permettere alla testa lessicale di **trasmettere** il valore dei propri tratti ai sintagmi funzionali che lo dominano direttamente: il DP ad esempio **erediterà** i tratti dalla testa nominale, completando attraverso l'operazione di unione gli eventuali tratti sottospecificati e verificando la compatibilità con quelli specificati.
  Il **tratto testa** è quindi un tratto fondamentale, che dovrà essere associato a determinati elementi lessicali e che determinerà la "direzionalità" dell'accordo.

Grammatiche ad unificazione e parsing - HPSG

(… fine)

- **Sottocategorizzazione**

  la sottocategorizzazione viene espressa attraverso un tratto **subcat** associato ad esempio al verbo:

$$\begin{bmatrix} \text{Ortografia} = \text{voglio} \\ \\ \text{Cat} = \text{V} \\ \\ \text{Testa} \begin{bmatrix} \text{Subcat} < [\text{Cat} = \text{NP}], \begin{bmatrix} \text{Cat} = \text{VP} \\ \text{Testa} = [\text{forma\_infinitiva}] \end{bmatrix} > \end{bmatrix} \end{bmatrix}$$

Implementazione dell'unificazione

- Il livello di **astrazione** della definizione permette di applicare l'unificazione a qualsiasi algoritmo
- Si possono distinguere almeno **due modalità di integrazione** sostanzialmente diverse:
  - **filtro** – si escludono le soluzione incompatibili una volta che tutte le soluzioni sono state generate dalla CFG
  - **vincolo** – constraints al parsing da valutare di volta in volta in modo da impedire immediatamente la generazione di strutture che poi alla fine non potrebbero essere unificate
- La seconda soluzione è spesso la migliore (se si riesce a minimizzare lo sforzo computazionale che i constraints richiedono)

Algoritmo di Earley + unificazione

(inizio …)

- Le sostanziali modifiche rispetto all'algoritmo originale sono essenzialmente quattro:
  a. le regole di riscrittura vengono integrate con **informazioni sui tratti**:
     S → DP VP <DP accordo = VP accordo>
  b. agli stati che rappresentano le regole parzialmente parseggiate vengono associate informazioni sui tratti in forma di **DAG**:
     S → • DP V, [0,0], [], DAG
  c. quando si applica l'**operazione di completamento**, il nuovo costituente che deve essere integrato, verrà valutato in base all'algoritmo di unificazione per verificarne la **compatibilità** con il DAG fino a quel momento costruito dall'applicazione della regola
  d. anche l'operazione di **incolonnamento** (che inseriva in coda agli stati solo regole non presenti nello stack) adesso dovrà verificare la compatibilità del DAG prima di escludere l'incolonnamento di regole che per la grammatica precedente potevano apparire identiche

Algoritmo di Earley + unificazione

(… fine)

- **COPYDAG**

  vista la natura dell'algoritmo di unificazione che essenzialmente tenderebbe a modificare i DAG compatibili, perderemmo l'utile proprietà dell'algoritmo di Earley di riutilizzare, senza rigenerarle, strutture già create durante l'applicazione delle regole.

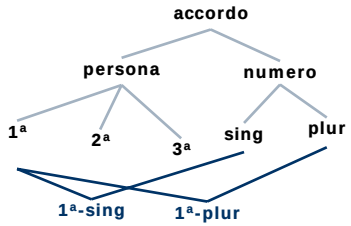
  In effetti se l'unificazione fallisse, ci ritroveremmo con una **struttura modificata** ed **inutilizzabile** per future operazioni. Per evitare questo si introduce un'operazione di COPYDAG, che ogni volta conserva una copia del DAG che la regola cerca di unificare.

Tipologie, ereditarietà... e problemi

- Problema dei **vincoli nell'assegnazione di valore**:
  niente vieta di assegnare valore *femminile* al tratto *numero*
- **Mancate generalizzazioni**:
  ad esempio ogni sottocategorizzazione verbale viene codificata indipendentemente per ogni singola entrata verbale, ma ci sarebbero ragionevoli generalizzazioni che si potrebbero fare (es. verbi intransitivi, inaccusativi ecc.)
- Per queste ragioni si introducono le **tipizzazioni**, cioè **classi di strutture di tratti** che rappresentano uniformemente elementi che hanno un comportamento linguistico significativamente omogeneo (es. caso, sottocategorizzazione verbale...); in particolare il sistema dei tipi permette di definire:
  - **condizioni di appropriatezza** che specificano quali tratti sono adeguati in un determinato tipo
  - **gerarchia di tipi**, attraverso cui i tipi più specifici ereditano le caratteristiche da quelli più astratti
  - **unificazione di tipi** oltre che di semplici tratti

Tipologie ed ereditarietà

- **esempio di gerarchia di tipi:**



- molti sforzi si sono concentrati sull'arricchimento delle proprietà espresse dai **tratti tipizzati**, tra queste l'idea dei
  - **valori di default** (valori assegnati ad un tratto quando questo rimane sottospecificato) o la
  - **priorità di unione** (viene specificato l'ordine di combinazione di certi tratti all'interno della gerarchia)

Relazioni a distanza

1. [Quali libri] [$_{VP}$ darai [$_{ogg.\ dir.}$ _] [$_{ogg.\ indir}$ a Maria]]?
2. [Quali libri] [$_{VP}$ hai detto [$_{ogg\ dir}$?]] che [$_{VP}$ darai [$_{ogg\ dir}$?] a Maria]?
3. [Chi] [$_{sogg}$?] [$_{VP}$ ha detto] che [$_{sogg}$?] [$_{VP}$ è andato via]?
4. [Quando] [$_{VP}$ hai detto [$_{mod}$?]] che [andrai da Maria [$_{mod}$?]]?

- **GAP List** (Sag & Wasow 1999)
  per gestire tali relazioni in modo da soddisfare anche "a distanza" **accordo** e **sottocategorizzazione** si tiene una lista degli elementi mancanti

- **FILLER**
  Elementi (es. [quali libri]) che possono entrare in una relazione di unificazione, dato un adeguato quadro di sottocategorizzazione, con gli elementi nella GAP list

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 trasparency** (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 trasparency** (Bresnan 78) quando alle varie proprietà grammaticali sono associate differenti regole nel modello computazionale che globalmente riproducono la struttura dell'ipotetico parser umano.
- 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.

Da regole a principi e parametri (lezione 2)

(inizio …)

Principle & Parameter Parsing

- **regole** — specifiche e valide per una sola lingua
- **principi & parametri** — universali linguistici + settaggio parametri di variazione
- Ricerca di una migliore **adeguatezza esplicativa** oltre che **descrittiva**
- **Obiettivo**: cogliere gli universali linguistici descrivendo precisamente la limitata variabilità sintattica
- I **principle-based parsers** (Barton 1984, Berwick e Fong 90, Stabler 92) si ispirano a queste idee:
  - i principi della grammatica sono **assiomi** per il parser
  - il parser è un **sistema deduttivo** che inferisce le espressioni grammaticali e la loro struttura partendo da tali assiomi

Da regole a principi e parametri (lezione 2)

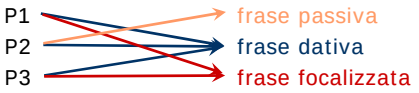
(… fine)

Principle & Parameter Parsing

- **regole**

  regola passivo → frase passiva
  regola dativo → frase dativa
  regola di focalizzazione → frase focalizzata
  …

- **principi & parametri**



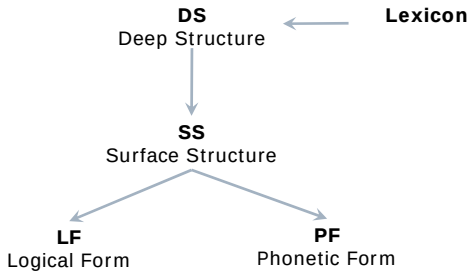
- potenzialmente una decina di principi + pochi parametri possono generare migliaia di regole

Principi e parametri (lezione 2)

(inizio …)

Principle & Parameter Parsing

- Modello a "T"

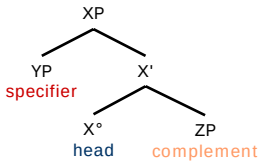


Principi e parametri (lezione 2)

(… continua …)

Principle & Parameter Parsing

- Alcuni principi
  - **X' theory**



  - **θ - criterion**

    ogni argomento deve ricevere uno ed un solo ruolo tematico (e ogni ruolo tematico è assegnato ad uno ed un solo argomento)

  - **Case filter**

    ogni NP lessicale deve ricevere un caso (P e V_{finito} assegnano caso)

Principi e parametri (lezione 2)

(… continua …)

Principle & Parameter Parsing

- Altri principi
  - **Move α**
    una categoria può muoversi in qualsiasi momento, ovunque
  - **Free indexation**
    indici sono liberamente assegnati alle categorie in posizione A(rgomentale)
  - **Binding theory**
    condizione A – Un'anafora (es. *se stessa*) è legata nel suo dominio di legamento (binding domain)
    condizione B – Un pronome (es *lei*) è libero nel suo dominio di legamento
    condizione C – Un'espressione referenziale (es. *Maria*) è sempre libera

Principi e parametri (lezione 2)

(… fine)

Principle & Parameter Parsing

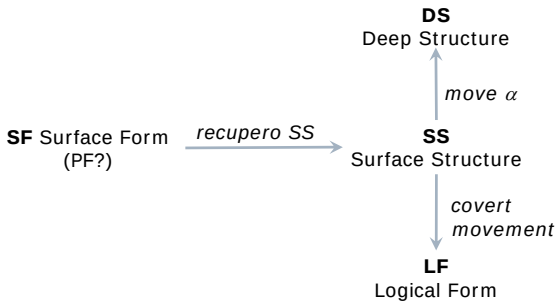
- **Generatori**
  principi che producono più strutture di quante non ne ricevano in input:
  - **Move α**
  - **Free indexation**
  - **…**
- **Filtri**
  principi che selezionano solo parte delle strutture che ricevono in input:
  - **X' theory**
  - **θ - criterion**
  - **Case filter**

PAPPI (Fong 1991)

(inizio …)

Principle & Parameter Parsing

- Specificare i principi grammaticali secondo un linguaggio di "alto livello" (logica del primo ordine) che poi vengono compilati nel parser (ottimo stratagemma per controllare l'obsolescenza delle teorie linguistiche)



PAPPI (Fong 1991)

(… continua …)

Principle & Parameter Parsing

- **Case Filter**
  - **definizione**
    "ogni NP lessicale deve ricevere un caso a SS"
  - **(pseudo) formalizzazione**
    $\forall$ NP: NP $\subset$ CF_{SS}, NP → caso
  - **implementazione in Prolog**

```
:- caseFilter in_all_configuration CF where
        lexicalNP(CF) then assignedCase(CF).

lexicalNP :- cat(NP, np), +\ ec(NP).
assignedCase(X) :- X has_features case(Case), assigned (Case).
```

PAPPI (Fong 1991)

(… continua …)

Operation	Encoded Using
θ-role assignment	in_all_configurations
θ-criterion	in_all_configurations
D-structure θ-condition	in_all_configurations
Subjacency	compositional_cases_on
Inherent Case assignment	in_all_configurations
Structural Case assignment	in_all_configurations
$\overline{\text{S}}$-deletion	in_all_configurations
Case filter	in_all_configurations
Case condition on traces	in_all_configurations
Trace theory (chain formation at S-structure)	compositional_cases_on
Wh-movement in syntax	in_all_configurations
Coindex Subject and INFL	in_all_configurations
Free Indexation	compositional_cases_on
Functional Determination	compositional_cases_on
Condition A	in_all_configurations (transformed)
Condition B	in_all_configurations (transformed)
Condition C	in_all_configurations (transformed)
Empty Category Principle (ECP)	in_all_configurations (transformed)
Control	compositional_cases_on
LF movement	Hand-coded
ECP at LF	in_all_configurations (transformed)
License operator-variables	in_all_configurations
License syntactic adjuncts	in_all_configurations
Wh-Comp requirement at LF	in_all_configurations

PAPPI (Fong 1991)

(… continua …)

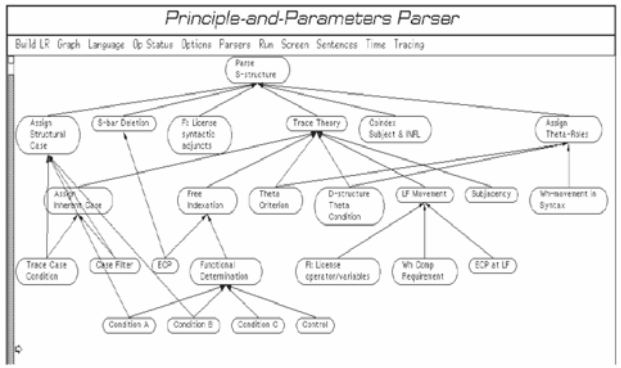


Figure 1.6 Operational dependency graph.

PAPPI (Fong 1991) – Ordinamento dei principi

(… fine)

frase di input → **PS** → **P_1** → **P_3** → … → **P_n** → LF

- **16 principi = 16! ordini possibili…**
  appena 20.922.789.888.000 possibilità (compito x casa: trovare quello migliore!!!)
- ogni ordine ha grossissime variazioni di performance rispetto ad una determinata frase in input
- non esiste un ordinamento universale che garantisca un'efficienza assoluta
- usare indizi (economici) per preordinare generatori e filtri

Strategie di controllo

- **Analysis-by-synthesis**
  produci il producibile, quindi testa le produzioni con il reale input (top-down parallelo)
- **Genera e testa**
  fai una scelta alla volta, produci una frase, quindi testa la produzione con il reale input (top-down seriale, depth first)
- **Coroutining, freezing, clause selection, ordering**
  riorganizza lo spazio del problema tentando sistemazioni alternative delle regole di produzione (principi in questo contesto)
- **Proprietà di covering**
  compila (off-line) la grammatica in modo da collassare insieme gruppi di principi riducendo ottimalmente il numero di passi da compiere

PAPPI (Fong 1991)

Principle & Parameter Parsing

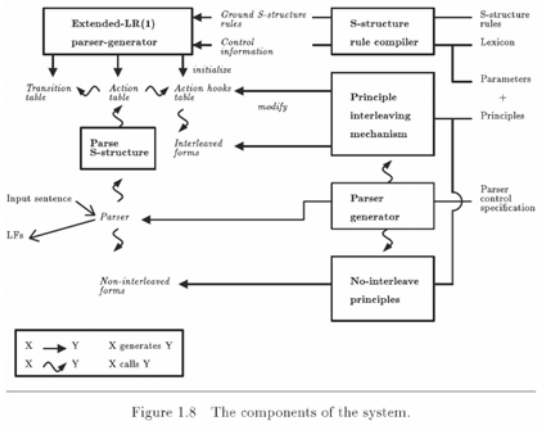


Figure 1.8 The components of the system.

La formalizzazione di grammatiche minimaliste

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 ∪ I) where
P are phonetic features and I are semantic ones;

Cat is a finite set of syntactic features,
Cat = (base ∪ select ∪ licensors ∪ licensees) where
base are standard categories {comp, tense, verb, noun …},
select specify a selection requirement {=x | x base}
licensees force phrasal movement {-wh, -case …},
licensors satisfy licensee requirements {+wh, +case …}

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};

La formalizzazione di grammatiche minimaliste

Grammatiche Minimaliste

Example of a toy Minimalist Grammar:

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}

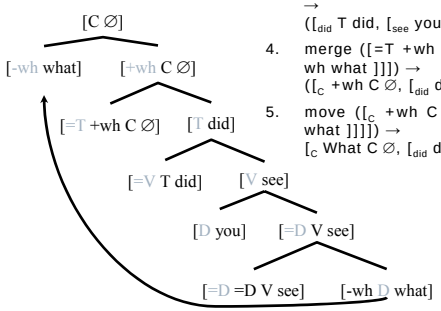
Lex = { [-wh D what], [=V T did], [D you], [=D =D V see], [=T +wh C ∅] }

F = {*merge*, *move*} such that:
merge (X, Y) = [$_X$ X Y]
(if and only if [=*F* X] and [*F* Y])
move (X, Y) = [[$_X$ Y X] W, t$_Y$]
(if [+*g* X] and [-*g* Y] with W possibly null, without any selecting/selector feature *g* in W)

La formalizzazione di grammatiche minimaliste

Grammatiche Minimaliste

1. merge ([=D =D V see], [-wh D what]) →
   [$_{see}$ =D V see, -wh what]
2. merge ([D you], [=D V see, -wh what]) →
   [$_{see}$ you, [$_{see}$ V see, -wh what]]
3. merge ([=V T did], [$_{see}$ you, [$_{see}$ V see, -wh what]]) →
   ([$_{did}$ T did, [$_{see}$ you, [$_{see}$ see, -wh what]]]
4. merge ([=T +wh C ∅], [$_{did}$ T did, [$_{see}$ you, [$_{see}$ see, -wh what]]]) →
   ([$_C$ +wh C ∅, [$_{did}$ did, [$_{see}$ you, [$_{see}$ see, -wh what]]]])
5. move ([$_C$ +wh C ∅, [$_{did}$ did, [$_{see}$ you, [$_{see}$ see, -wh what]]]]) →
   [$_C$ What C ∅, [$_{did}$ did, [$_{see}$ you, [$_{see}$ see, t$_{what}$]]]]



Prossima lezione

(Mercoledì 21 Marzo, ore 16:30-18:30, Aula G, Palazzo S. Galgano, Via Roma 47

- Laboratorio
  - Parsing e Transfer
    - Costruzione di grammatiche
    - Programmazione algoritmi in PHP
    - Grammatiche e transfer