Seminari del Corso di Linguistica Computazionale

Siena 19 maggio 2003

Parsing Sintattico con Context Free Grammars

Michelangelo Falco - falco@f2o.org

Sommario

- Sintassi e Context Free Grammars (CFG)
- Problemi delle CFG
- Parsing con CFG: top-down, buttom-up
- Problemi del parsing
- Algoritmo di Earley

Sintassi

- Con sintassi si intende la nostra conoscenza implicita della **struttura** della lingua materna che abbiamo perfettamente acquisito nei primi tre o quattro anni di vita senza nessuna istruzione esplicita
- Non le regole che ci sono state insegnate a scuola

Sintassi

- Perché ci interessiamo di sintassi in linguistica computazionale?
  - Correttori ortografici
  - Importante per l'analisi semantica e quindi:
    - Question answering
    - Information extraction
    - Traduzione automatica
  - Versioni stocastiche sono state incorporate in speech recognizers

Context-Free Grammars

- Catturano costituenza e ordine
  - L'ordine è semplice
    Esprimono le regole che governano l'ordine della parole e di unità più grandi nel linguaggio
  - Cos'è la costituenza?
    Come le parole si raggruppano in unità e come i vari tipi di unità si comportano.
  - Costituenti e la struttura del linguaggio
  - Test di costituenza: coordinazione, pronominalizzazione, frasi scisse etc..

Esempi di CFG

- S -> NP VP
- NP -> Det NOMINAL
- NOMINAL -> Noun
- VP -> Verb
- Det -> *a*
- Noun -> *flight*
- Verb -> *left*

CFGs

- S -> NP VP
  - Questo ci dice che in questo linguaggio c'è un'unità chiamata S, una NP e una VP
  - Che S consiste di un NP immediatamente seguito da un VP
  - Non dice che questo è l'unico tipo di S
  - Né dice che questo è l'unico luogo in cui NP e VP occorrono

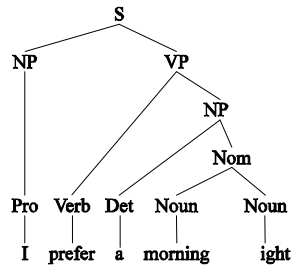
Generatività

- Queste regole possono essere viste come meccanismi di analisi o di generazione
  - Generano stringhe nel linguaggio
  - Rifiutano stringhe non nel linguaggio
  - Forniscono strutture (alberi) per le stringhe nel linguaggio

Derivazioni

- Una derivazione è una sequenza di regole applicate a una stringa che rende conto della stringa stessa
  - Copre tutti gli elementi nella stringa
  - Copre solo gli elementi nella stringa

Derivazioni come alberi



Parsing

- Il parsing è il processo che prende una stringa e una grammatica come input e restituisce uno o più alberi di parsing per quella stringa

Context?

- La nozione di context in CFG significa semplicemente che i nodi non terminali nella parte sinistra di una regola sono presenti per sé stessi, indipendentemente dal contesto (vi sono grammatiche non Context Free)
  - A -> B C
  - Significa che si può riscrivere A come B seguito da C indipendentemente dal contesto in cui si trova A

Costituenti principali (Inglese)

- Sentences
- Noun phrases
- Verb phrases
- Prepositional phrases

Tipi di frase

- Dichiarative: A plane left
  S -> NP VP
- Imperative: Leave!
  S -> VP
- Domande Si-No: Did the plane leave?
  S -> Aux NP VP
- Domande WH: When did the plane leave?
  S -> WH Aux NP VP

Ricorsione

- Avremo a che fare con regole come le seguenti in cui il nodo non terminale sulla sinistra appare anche da qualche parte sulla destra (direttamente)
  NP -> NP PP [[The flight] [to Boston]]
  VP -> VP PP [[departed Miami] [at noon]]

Ricorsione

- E' proprio questa proprietà che rende la sintassi interessante (creatività!)
  - flights from Denver
  - Flights from Denver to Miami
  - Flights from Denver to Miami in February
  - Flights from Denver to Miami in February on a Friday
  - Flights from Denver to Miami in February on a Friday under $300
  - Flights from Denver to Miami in February on a Friday under $300 with lunch

Ricorsione

- E' proprio questa proprietà che rende la sintassi interessante
  [[flights] [from Denver]]
  [[[Flights] [from Denver]] [to Miami]]
  [[[[Flights] [from Denver]] [to Miami]] [in February]]
  [[[[[Flights] [from Denver]] [to Miami]] [in February]] [on a Friday]]
  Etc.

Il punto

- Se si ha una regola come
  - VP -> V NP
  - Importa solamente che ciò che si trova dopo il verbo sia un NP. Non importa la struttura interna dell'NP

Il punto

- VP -> V NP
- I hate
  flights from Denver
  Flights from Denver to Miami
  Flights from Denver to Miami in February
  Flights from Denver to Miami in February on a Friday
  Flights from Denver to Miami in February on a Friday under $300
  Flights from Denver to Miami in February on a Friday under $300 with lunch

Congiunzione

- S -> S and S
  - John went to NY and Mary followed him
- NP -> NP and NP
- VP -> VP and VP
- …
- La regole corretta è
  X -> X and X

Problemi

- Accordo
- Sottocategorizzazione
- Movimento

Accordo

- This dog
- Those dogs
- *This dogs
- *Those dog

- This dog eats
- Those dogs eat
- *This dog eat
- *Those dogs eats

Accordo

- In Inglese,
  - Soggetti e verbi devono accordarsi in persona e numero
  - Determinanti e nomi devono accordarsi in numero
- Molte lingue, fra cui l'italiano, hanno sistemi di accordo molto più complessi.

Possibile soluzione con CFG

- S -> NP VP
- NP -> Det Nominal
- VP -> V NP
- …
- SgS -> SgNP SgVP
- PlS -> PlNp PlVP
- SgNP -> SgDet SgNom
- PlNP -> PlDet PlNom
- PlVP -> PlV NP
- SgVP ->SgV Np
- …

Soluzioni CFG per l'Accordo

- Funziona senza bisogno di ricorrere a formalismi più potenti
- Ma non è affatto elegante

Sottocategorizzazione

- Sneeze: John sneezed
- Find: Please find [a flight to NY]$_{NP}$
- Give: Give [me]$_{NP}$[a cheaper fare]$_{NP}$
- Help: Can you help [me]$_{NP}$[with a flight]$_{PP}$
- Prefer: I prefer [to leave earlier]$_{TO\text{-}VP}$
- Told: I was told [United has a flight]$_{S}$
- …

Sottocategorizzazione

- *John sneezed the book
- *I prefer United has a flight
- *Give with a flight

- La sottocategorizzazione esprime le restrizioni che un predicato impone sul numero e sul tipo di argomenti che prende

Quindi?

- Quindi le varie regole per i VP generano troppo.
  - Permettono la presenza di stringhe che contengono verbi e argomenti che non stanno insieme
  - Per esempio
  - VP -> V NP perciò
    Sneezed the book è un VP ben formato poiché "sneeze" è un verbo e "the book" è un NP ben formato

Soluzioni CFG per la sottocategorizzazione

- VP -> V
- VP -> V NP
- VP -> V NP PP
- …

- VP -> IntransV
- VP -> TransV NP
- VP -> TransPP NP PP
- …

Movimento

- Esempio di base
  - [[My travel agent]$_{NP}$ [booked [the flight]$_{NP}$]$_{VP}$]$_{S}$
- "book" è un verbo transitivo. Richiede un singolo NP arg all'interno del VP come argomento e un singolo NP soggetto.

Movimento

- Esempio con movimento
  - Which flight do you want me to have the travel agent book?
- L'oggetto diretto di "book" non appare nella posizione corretta, è lontano dalla posizione in cui dovrebbe apparire.
- Si noti che è separato dal suo verbo da altri due verbi

Il punto

- CFGs sembrano essere un formalismo che rende conto di molta della struttura sintattica di base dell'inglese
- Ma vi sono problemi
  - Che possono essere affrontati adeguatamente, anche se non elegantemente, restando all'interno del quadro CFG
- Ci sono soluzioni più semplici e più eleganti che ci portano al di fuori delle CFGs (al di là del loro potere formale)

Parsing

- Il parsing con CFGs si riferisce al compito di assegnare alberi corretti alle stringhe di input usando regole context free
- Un albero è corretto se copre tutti e solo gli elementi dell'input e ha una S come radice
- Non significa che il sistema può scegliere l'albero corretto fra i possibili alberi

Parsing

- Il parsing implica una ricerca che richiede della scelte
- Cominciamo con qualche metodo di base:
  - top-down
  - bottom-up
  - top-down con filtro bottom-up

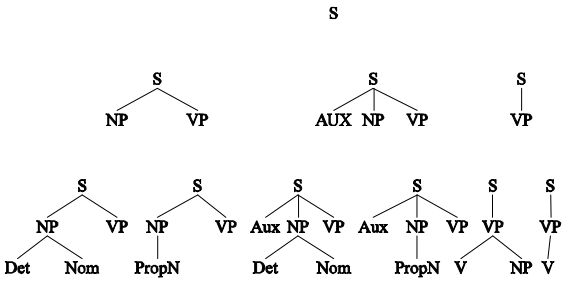
Si assuma che…

- Abbiamo tutte le parole già in un buffer
- L'input non è Part of Speech tagged
- Non ci curiamo dell'analisi morfologica
- Tutte le parole sono conosciute

Top-Down Parsing

- Poiché stiamo cercando di trovare alberi che hanno come radice una S (Sentences) cominciamo con regole che ci danno una S
- Quindi cerchiamo di scendere giù fino alle parole

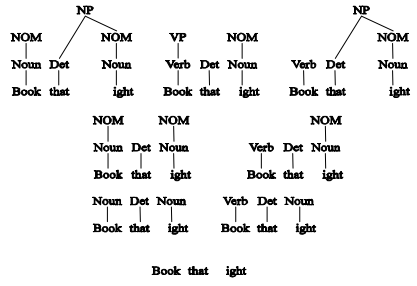
Spazio Top-Down



Bottom-Up Parsing

- Ovviamente, vogliamo anche alberi che coprano le parole di input. Quindi cominciamo con alberi che si legano alle parole in modo corretto
- Quindi cerchiamo si salire su dalle parole

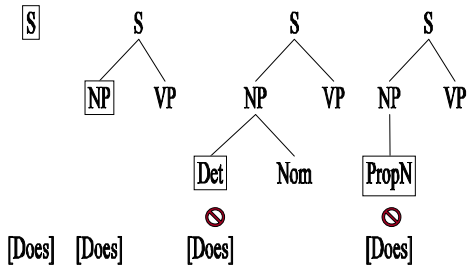
Spazio Bottom-Up



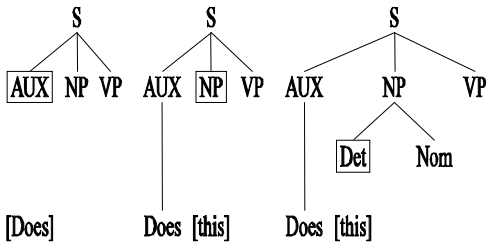
Controllo

- Abbiamo omesso di specificare come seguire gli spazi e come prendere le decisioni
  - Quale nodo proviamo a espandere al passo successivo?
  - Quale regola grammaticale si usa per espandere un nodo?

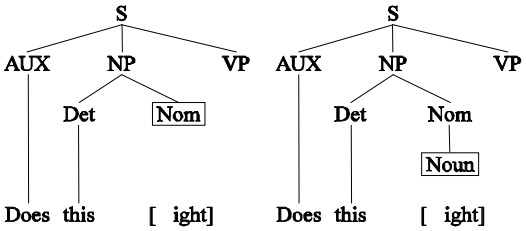
Ricerca Top-Down, Depth-First, Left-to-Right



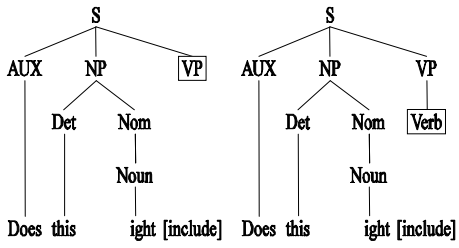
Esempio



Esempio

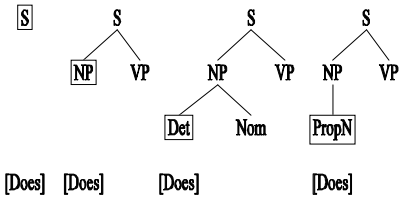


Esempio



Controllo

- Ha senso questa sequenza?



Top-Down e Bottom-Up

- Top-down
  - Forma solo alberi che possono essere le risposte
  - Ma suggerisce alberi che non sono consistenti con le parole
- Bottom-up
  - Forma solo alberi consistenti con le parole
  - Suggerisce alberi che non hanno senso globalmente

Quindi Combiniamoli

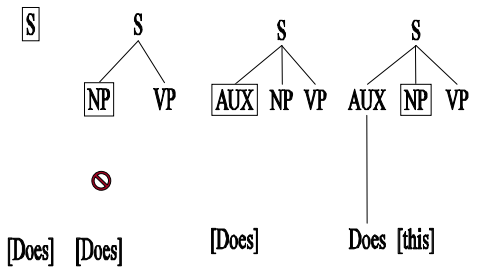
- Ci sono molti modi di combinare previsioni top-down con dati bottom-up per ottenere ricerche più efficienti
- La maggior parte usano un tipo come meccanismo di controllo e l'altro come filtro

  Per esempio nel parsing top-down con filtro bottom-up

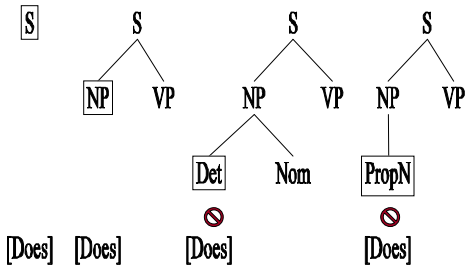
Parsing top-down con filtro bottom-up

- In un parser top-down, depth-first, left-to-right la parola corrente di input (left corner) deve servire come prima parola nella derivazione del nodo non espanso che si sta processando
- Perciò non si deve considerare ogni regola grammaticale se l'input corrente non può funzionare come prima parola lungo il lato sinistro di qualche derivazione
- Il filtro può essere implementato con tabelle che elencano tutte le categorie valide per il left-corner per ogni nodo non terminale

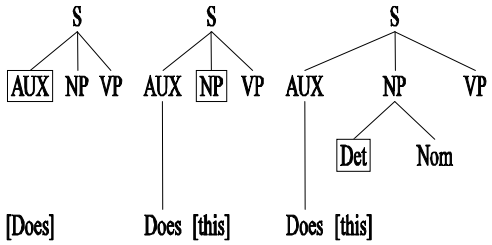
Bottom-Up Filtering



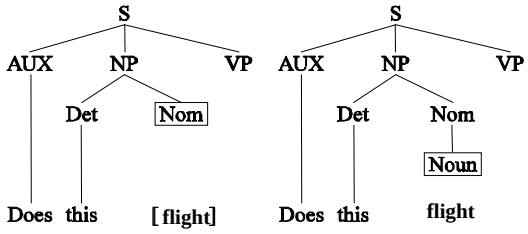
Top-Down, Depth-First, Left-to-Right Search



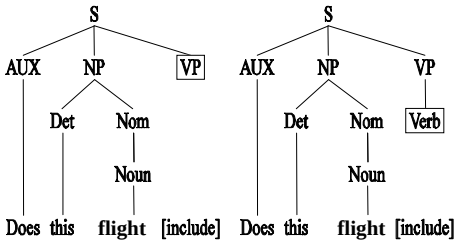
Esempio



Esempio



Esempio



Problemi

- Ricorsione a sinistra
- Ambiguità
- Parsing inefficiente dei subtrees

Left-Recursion

- Cosa accade nella situazione seguente?
  - S -> NP VP
  - S -> Aux NP VP
  - NP -> NP PP
  - NP -> Det Nominal
  - …
  - Con una frase che comincia con
    - Did the flight …
- Con grammatiche ricorsive a sinistra la ricerca depth-first può scendere ricorsivamente lungo un percorso e non tornare più a visitare gli stati non espansi

Ordinamento delle regole

- S -> Aux NP VP
- S -> NP VP
- NP -> Det Nominal
- NP -> NP PP

- La chiave per gli NP è applicare la regola ricorsiva dopo ogni regola non ricorsiva.
- Oppure è possibile creare una nuova grammatica weakly equivalent alla grammatica ricorsiva a sinistra, ma senza questa caratteristica

Ambiguità

- Ambiguità strutturale: la grammatica assegna spessissimo molte strutture a una stessa frase
  - ambiguità di attaccamento:
    Es: I shoot an elephant in my pajamas
  - ambiguità di coordinazione.
    old men and women

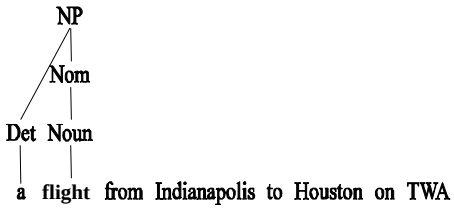
…

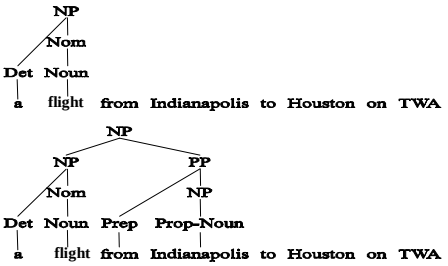
Ambiguità

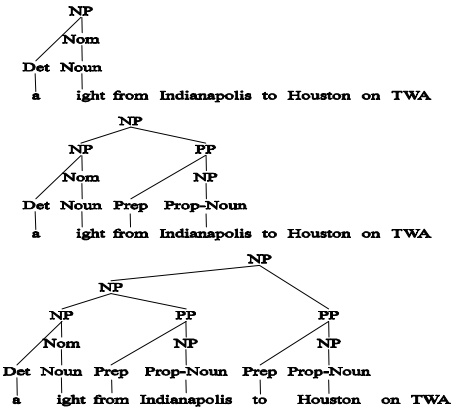
- Tutte queste ambiguità si combinano in modi complessi
- A causa dell'ambiguità il parsing richiede disambiguazione: scegliere il parsing corretto da una moltitudine di possibili alberi (gli algoritmi di disambiguazione richiedono conoscenza statistica e semantica)
- I parser senza algoritmi di disambiguazione restituiscono tutti i possibili alberi

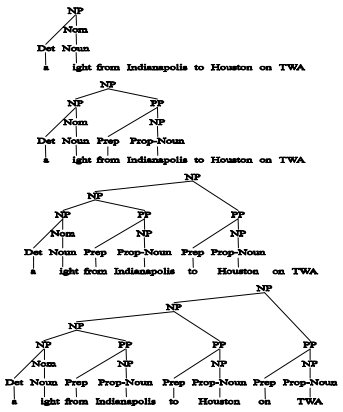
Parsing Ripetuto di Subtrees

- Il parser costruisce alberi validi per porzioni di input che sono poi scartati nel processo di backtracking. Scoprire poi che devono essere ricostruiti.
- Vediamo un esempio top-down, tenendo presente che gli stessi problemi si presentano per il parsing bottom-up









Dynamic Programming

- **Abbiamo bisogno di un metodo che riempia una tabella con risultati parziali che**
  - **Non ripeta il lavoro già fatto che si può evitare**
  - **Non cada nella ricorsione a sinistra**
  - **Risolva un problema esponenziale in un tempo polinomiale**

Earley Parsing

- Riempie una tabella con un singolo passaggio sulle parole di input
  - La tabella è di lunghezza N+1; N è il numero di parole
  - Le caselle della tabella rapprensentano
    - Costituenti completati e le loro posizioni
    - Costituenti in formazione
    - Costituenti predetti

Stati

- Le caselle sono chiamate stati e sono rappresentate con dotted-rules

S -> · VP	E' predetto un VP
NP -> Det · Nominal	Un NP è in formazione
VP -> V NP ·	E' stato trovato un VP

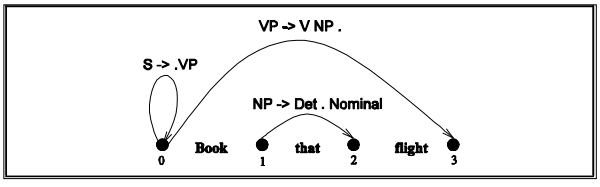
Stati/Posizioni

- Occorre sapere dove ci troviamo nell'input
  - book that flight

S -> · VP [0,0]	Un VP è predetto all'inizio della frase
NP -> Det · Nominal [1,2]	Un NP è in formazione; Il Det va da 1 a 2
VP -> V NP · [0,3]	E' stato trovato un DP che comincia a 0 e termina a 3

Graficamente

Grafo diretto aciclico delle tre dotted-rules sopra



Earley

- Come con la maggior parte degli approcci di programmazione dinamica, la risposta al problema si trova guardando nella tabella nelle posizioni giuste.
- In questo caso ci deve essere uno stato S nella colonna finale che vada da 0 a n+1 e sia completo.
- Se questo è il caso
  - S –> α · [0,n+1]

Earley

- Operazioni fondamentali dell'algoritmo, scorrendo in avanti attraverso la tabella da 0 a n+1
  - Nuovi stati predetti sono creati con le regole
  - Nuovi stati incompleti sono creati facendo avanzare gli stati esistenti man mano che sono scoperti nuovi costituenti
  - Nuovi stati completi sono creati allo stesso modo

Earley: Predictor

- Predictor: predice tutti gli stati che rappresentano aspettative top-down generate nel processo di parsing.
- Si applica a ogni stato che ha un nodo non terminale alla destra del dot che non è una part-of-speech.
- I nuovi stati sono inseriti nella stessy entry della tabella dello stato che li ha generati.
- Cominciano e finiscono nel punto dell'input dove gli stati generatori finiscono.
- Esempio:

  Applicato a S -> · VP [0,0] produce:

  VP -> · Verb, [0,0] e VP -> · Verb NP, [0,0] nella prima entry della tabella

Earley: Scanner

- Quando uno stato ha una part-of-speech alla destra del dot, lo scanner esamina l'input e incorpora uno stato corrispondente alla part-of-speech predetta nella tabella.
- Si crea un nuovo stato a partire dallo stato input con il dot avanzato oltre la categoria di input predetta
- Esempio

  Quando lo stato VP -> · Verb NP, [0,0] è prodotto, lo scanner consulta la parola corrente nell'input perché Verb è una part-of-speech. Book può essere un verbo e quindi corrisponde all'aspettativa nello stato corrente. Si crea allora il nuovo stato VP -> Verb · NP, [0,1]. Il nuovo stato è aggiunto al campo della tabella che segue quella che si sta processando.

Earley: Completer

- Completer è applicato a uno stato quando il suo dot ha raggiunto la parte finale di una regola
- Intuitivamente la presenza di un tale stato rappresenta il fatto che il parser ha scoperto una particolare categoria grammaticale che si estende sull'input
- Lo scopo di completer è trovare e portare avanti tutti gli stati precedentemente creati che cercavano la categoria grammaticale trovata nella posizione corrente nell'input. I nuovi stati sono creati copiando i vecchi e spostando in avanti il dot al di là della categoria attesa e inserendo il nuovo stato nel campo corrente della tabella
- Esempio
  Quando l'algoritmo processa lo stato NP -> Det Nominal ·, [1,3] completer cerca gli stati che terminano a 1 e che aspettano un NP. Trova lo stato VP -> Verb · NP, [0,1] creato da scanner. Quindi si aggiunge il nuovo stato completo VP -> Verb NP · , [0,3]

Esempio

- Book that flight
- Dovremmo trovare un S da 0 a 3 che è una stato completo

Example

La tabella si alimenta con le predizione top-down per S e transitivamente per tutti i left-corners di questi alberi

Chart[0]

γ ■ ▪ *S*	[0,0]	Dummy start state
S ■ ▪ *NP VP*	[0,0]	Predictor
NP ■ ▪ *Det NOMINAL*	[0,0]	Predictor
NP ■ ▪ *Proper-Noun*	[0,0]	Predictor
S ■ ▪ *Aux NP VP*	[0,0]	Predictor
S ■ ▪ *VP*	[0,0]	Predictor
VP ■ ▪ *Verb*	[0,0]	Predictor
VP ■ ▪ *Verb NP*	[0,0]	Predictor

Example

Chart[0]

γ ■ ▪ *S*	[0,0]	Dummy start state
S ■ ▪ *NP VP*	[0,0]	Predictor
NP ■ ▪ *Det NOMINAL*	[0,0]	Predictor
NP ■ ▪ *Proper-Noun*	[0,0]	Predictor
S ■ ▪ *Aux NP VP*	[0,0]	Predictor
S ■ ▪ *VP*	[0,0]	Predictor
VP ■ ▪ *Verb*	[0,0]	Predictor
VP ■ ▪ *Verb NP*	[0,0]	Predictor

Chart[1]

Verb ■ *book* ▪	[0,1]	Scanner
VP ■ *Verb* ▪	[0,1]	Completer
S ■ *VP* ▪	[0,1]	Completer
VP ■ *Verb* ▪ *NP*	[0,1]	Completer
NP ■ ▪ *Det NOMINAL*	[1,1]	Predictor
NP ■ ▪ *Proper-Noun*	[1,1]	Predictor

Example

Chart[1]

Verb → *book* •	[0,1]	Scanner
VP → *Verb* •	[0,1]	Completer
S → *VP* •	[0,1]	Completer
VP → *Verb* • *NP*	[0,1]	Completer
NP → • *Det NOMINAL*	[1,1]	Predictor
NP → • *Proper-Noun*	[1,1]	Predictor

Chart[2]

Det → *that* •	[1,2]	Scanner
NP → *Det* • *NOMINAL*	[1,2]	Completer
NOMINAL → • *Noun*	[2,2]	Predictor
NOMINAL → • *Noun NOMINAL*	[2,2]	Predictor

Example

Chart[2]

Det → *that* •	[1,2]	Scanner
NP → *Det* • *NOMINAL*	[1,2]	Completer
NOMINAL → • *Noun*	[2,2]	Predictor
NOMINAL → • *Noun NOMINAL*	[2,2]	Predictor

Chart[3]

Noun → *ight* •	[2,3]	Scanner
NOMINAL → *Noun* •	[2,3]	Completer
NOMINAL → *Noun* • *NOMINAL*	[2,3]	Completer
NP → *Det NOMINAL* •	[1,3]	Completer
VP → *Verb NP* •	[0,3]	Completer
S → *VP* •	[0,3]	Completer
NOMINAL → • *Noun*	[3,3]	Predictor
NOMINAL → • *Noun NOMINAL*	[3,3]	Predictor

Cos'è Earley?

- Che tipo di parser abbiamo appena descritto?
  - Il parser di Earley … certo
  - Non è un parser ma un riconoscitore
    - La presenza di uno stato S con gli attributi giusti al posto giusto indica un riconoscimento
    - Ma non abbiamo un albero di parsing… quindi non abbiamo un parser

Earley: Recuperare Alberi

- Aggiungendo dei puntatori per recuperare la strutture degli S riconosciuti abbiamo un parser
- Tutti i possibili parsing per un input sono già nella tabella creata
- Dobbiamo solamente leggere tutti i puntatori all'indietro da ogni S completato nell'ultima colonna della tabella
- Implementazione: completer aggiunge un puntatore al vecchio stato in un nuovo campo della tebella per ogni nuovo stato

Earley e Ricorsione a Sinistra

- Earley risolve il problema della ricorsione a sinistra senza dover alterare la grammatica o limitare artificialmente la ricerca
  - Non reinserisce mai uno stato già presente nella tabella
  - Copia gli stati prima di portarli avanti

Earley e Ricorsione a Sinistra

- S -> NP VP
- NP -> NP PP

- La prima regola predice
  - S -> · NP VP [0,0] che aggiunge
  - NP -> · NP PP [0,0]
  - L'algoritmo si ferma a questo punto perché aggiungere qualunque predizione successiva sarebbe inutile

Features e Unificazione

- Problema dell'overgeneration delle CFGs
  - Accordo
  - Sottocategorizzazione
- Features structures: le categorie grammaticali e le regole grammaticali devono essere pensate come oggetti che possono avere insiemi di proprietà
- Operatore di Unificazione: operazione di base sulle features structures

  S -> NP VP

  solo se il numero dell'NP è uguale al numero del VP

CFGs Probabilistiche

- Problema dell'ambiguità e la disambiguazione
- Modello probabilistico
  - Si assegnano probabilità agli alberi di parsing
- Occorrono le probabilità per il modello
- Parsing con le probabilità
  - Leggera modifica all'approccio dynamic programming (algoritmo CYK)
  - Il compito è trovare l'albero con la probabilità massima dato un input

Grazie per l'attenzione

Michelangelo Falco
falco@f2o.org

Le slides saranno disponibili in rete:
http://falco.f2o.org/academia/