
Tecnologie del Linguaggio Naturale

Parte Prima

Lezione n. 04 (3)

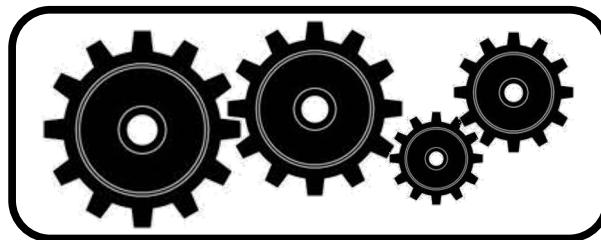
Sintassi: grammatiche e

parser a dipendenze

19-03-2021

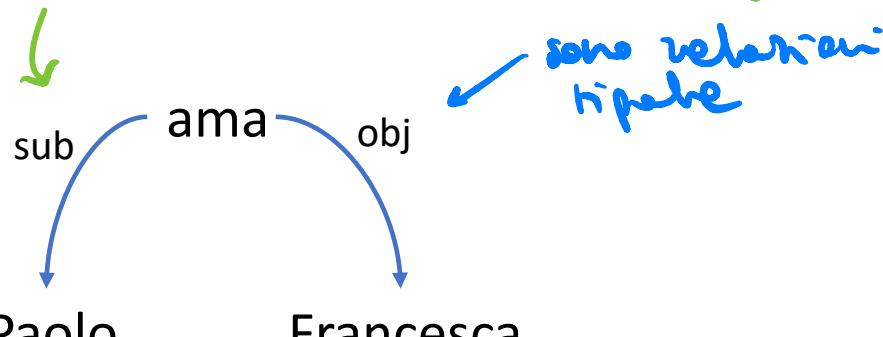
Parser a dipendenze

Paolo ama Francesca



Paolo è il soggetto delle parole *ama*

ama è in relaz. con
Paolo, di tipo *legg.*



Outline

- Sintassi a dipendenze
- Parsing a dipendenze: approcci
- Parsing deterministico a “transizioni”
- Parsing a regole per ~~dipendenze~~ a vincoli

Outline

- Sintassi a dipendenze
- Parsing a dipendenze: approcci
- Parsing deterministico a “transizioni”
- Parsing a regole per dipendenze a vincoli

Lucien Tesnière (1959)

The sentence is an organized whole, the constituent elements of which are words. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence.

The structural connections establish dependency relations between the words. Each connection in principle unites a superior term and an inferior term. The superior term receives the name governor. The inferior term receives the name subordinate. Thus, in the sentence *Alfred parle, parle* is the governor and *Alfred* the subordinate.

Lucien Tesnière (1959)

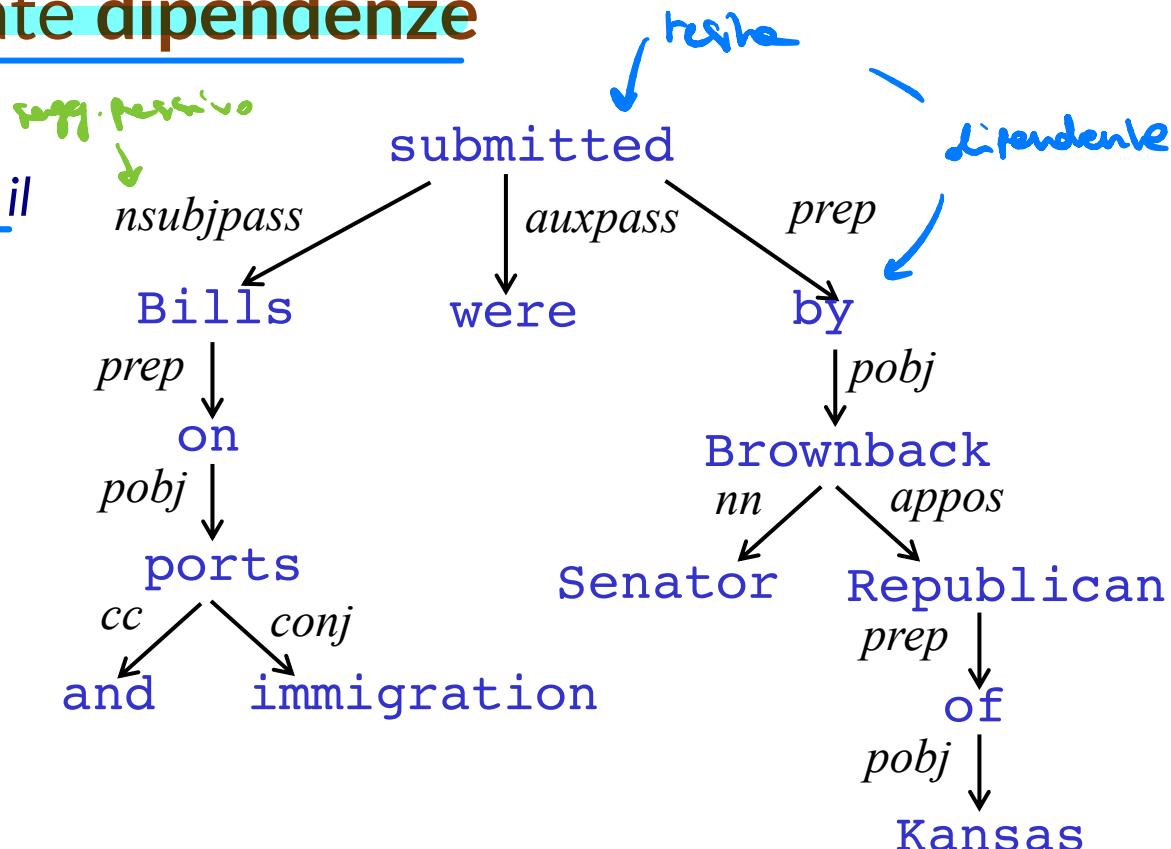
The sentence is an organized whole, the constituent elements of which are words. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. **Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence.**

The structural connections establish dependency relations between the words. Each connection in principle unites a superior term and an inferior term. The superior term receives the name **governor**. The inferior term receives the name **subordinate**. Thus, in the sentence **Alfred parle, parle** is the governor and **Alfred** the subordinate.

Sintassi a dipendenze

La sintassi a dipendenze postula che la struttura sintattica consiste di elementi lessicali connessi da relazioni binarie asimmetriche (graficamente frecce) chiamate **dipendenze**

Le dipendenze sono normalmente tipate con il nome di una relazione grammaticale (subject, prepositional object, apposition, etc.)



Cosa è una testa?

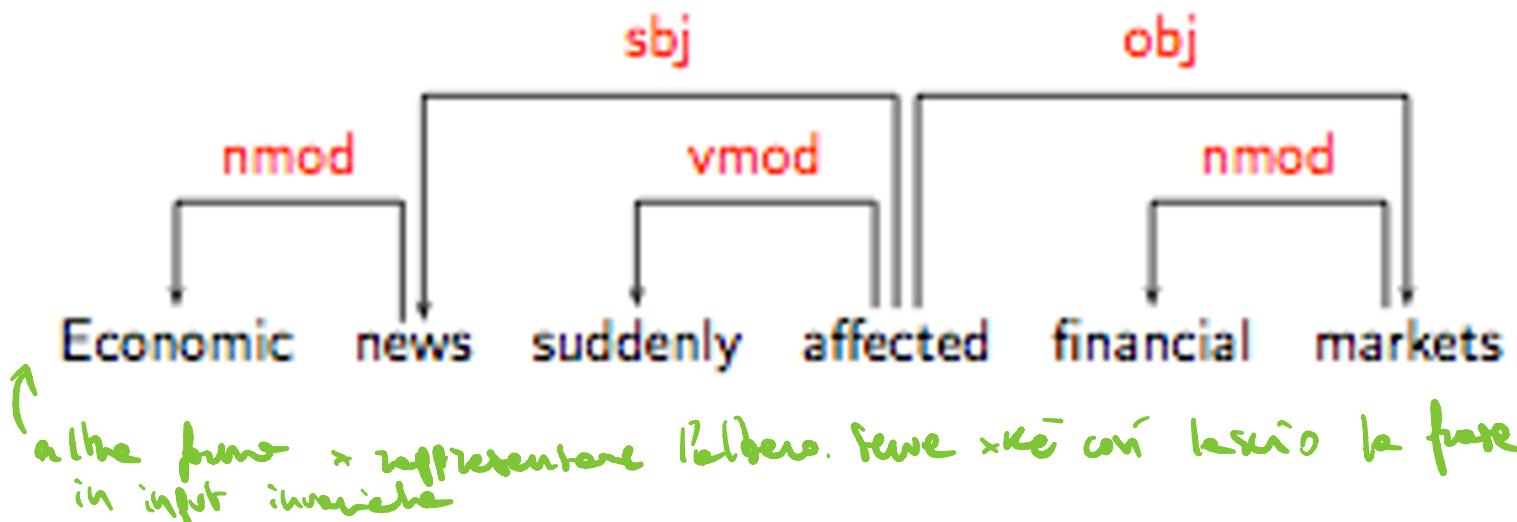
• Criteri sintattici per distinguere una head H e un dependent D in una costruzione C [Zwicky 1985, Hudson 1990]:

- H determina la categoria sintattica di C; H può sostituire C
- H è obbligatoria; D può essere opzionale
- H seleziona D e determina quando D è obbligatoria
- La forma di D dipende da H (agreement)
- La posizione nella frase di D è specificabile con riferimento a H.
- H determina la categoria semantica di C

Criteri: morfologici - sintattici - semantici

Teste evidenti

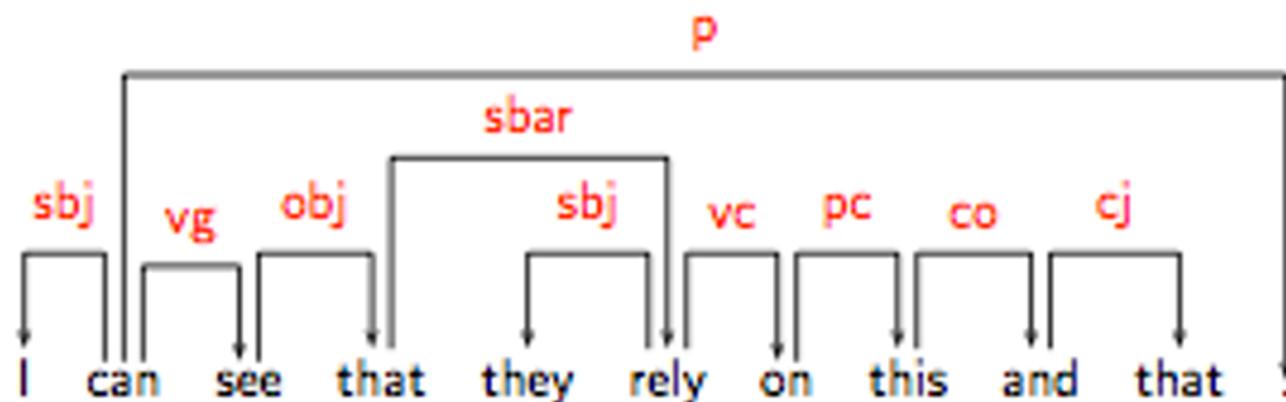
<u>Head</u>	<u>Dependent</u>
Verb	Subject (sbj)
Verb	Object (obj)
Verb	Adverbial (vmod)
Verb	Attribute (nmod)



Teste problematiche

non c'è sempre unico chi è
cora, di per sé dalle regole
che si leggono

Construction	<u>Head?</u>	<u>Dependent?</u>
Complex verb groups	auxiliary	main verb
Subordinate clauses	complementizer	verb
Coordination	coordinator	conjuncts
Prepositional phrases	preposition	nominals
Punctuation	verb	punct



2 approssimazioni sulla sintassi (Tesnière)

- bottiglia di vino -> di è una specie di funzione che
trasforma il nome in un aggettivo → "bottiglia vinate"
- cani e gatti -> la relazione è simmetrica

Dependency Grammar (2019) de Marneffe and Joakim Nivre

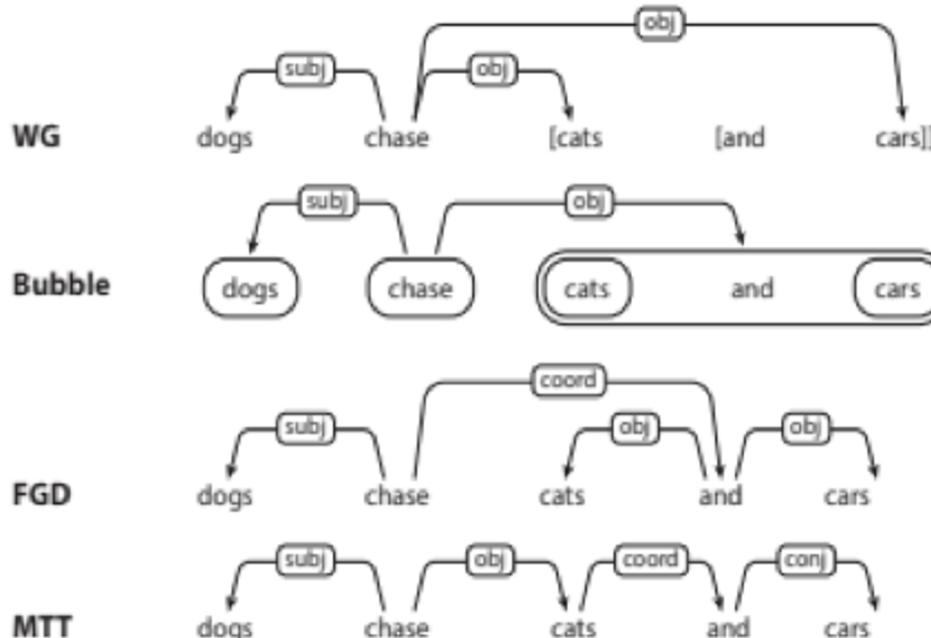


Figure 2

Treatment of coordination in four dependency frameworks: Word Grammar (WG), bubble tree (Bubble), Functional Generative Description (FGD), and Meaning-Text Theory (MTT).

Dependency Grammar (2019)

de Marneffe and Joakim Nivre

le grammatiche a componenti si basano sulla gramm. generative di Chomsky ("le quattro")

In formal and generative linguistics, this term (grammar) is often understood in the sense of a formal declarative system capable of generating languages and making predictions about grammaticality. Most frameworks in the dependency grammar tradition do not make use of grammars in this sense but are better described as syntactic analysis schemes that may be more or less formalized. These characteristics carry over to most computational models for dependency parsing, which rely on machine learning techniques to learn statistical regularities from annotated corpora without inducing a formal grammar.

*ci danno delle euristiche - per costruire
l'albero*

Dependency Grammar (2019)

de Marneffe and Joakim Nivre

↳ Vantaggi delle dipendenze:

- ◉ Generalization Across Languages
- ◉ Operationalization of Human Sentence Processing

Facts

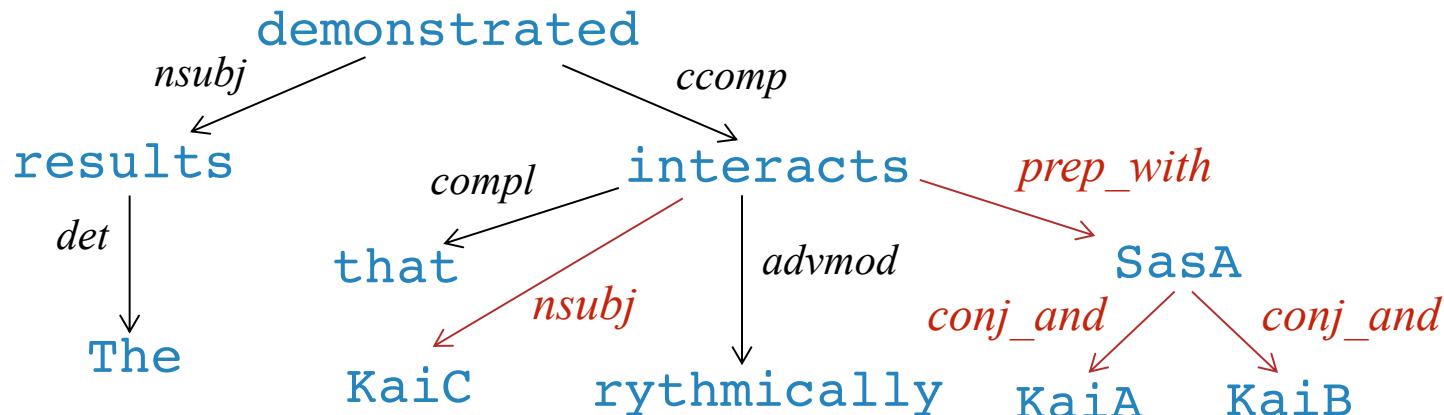
- ◉ Transparency and Simplicity of the Representation

Formalizzazione della sintassi a dipendenze

- Word Grammar (WG) [Hudson 1984, Hudson 1990]
- Functional Generative Description (FGD) [Sgall et al. 1986]
- Dependency Unification Grammar (DUG) [Hellwig 1986, Hellwig 2003]
- Meaning-Text Theory (MTT) [Melcuk 1988]
- (Weighted) Constraint Dependency Grammar ([W]CDG)
 - [Maruyama 1990, Harper and Helzerman 1995,
 - Menzel and Schröder 1998, Schroder 2002]
- Functional Dependency Grammar (FDG) [Tapanainen and Jarvinen 1997, Jarvinen and Tapanainen 1998]
- Topological/Extensible Dependency Grammar ([T/X]DG) [Duchier and Debusmann 2001, Debusmann et al. 2004]

molte

Information extraction e dipendenze: un esempio sulle proteine



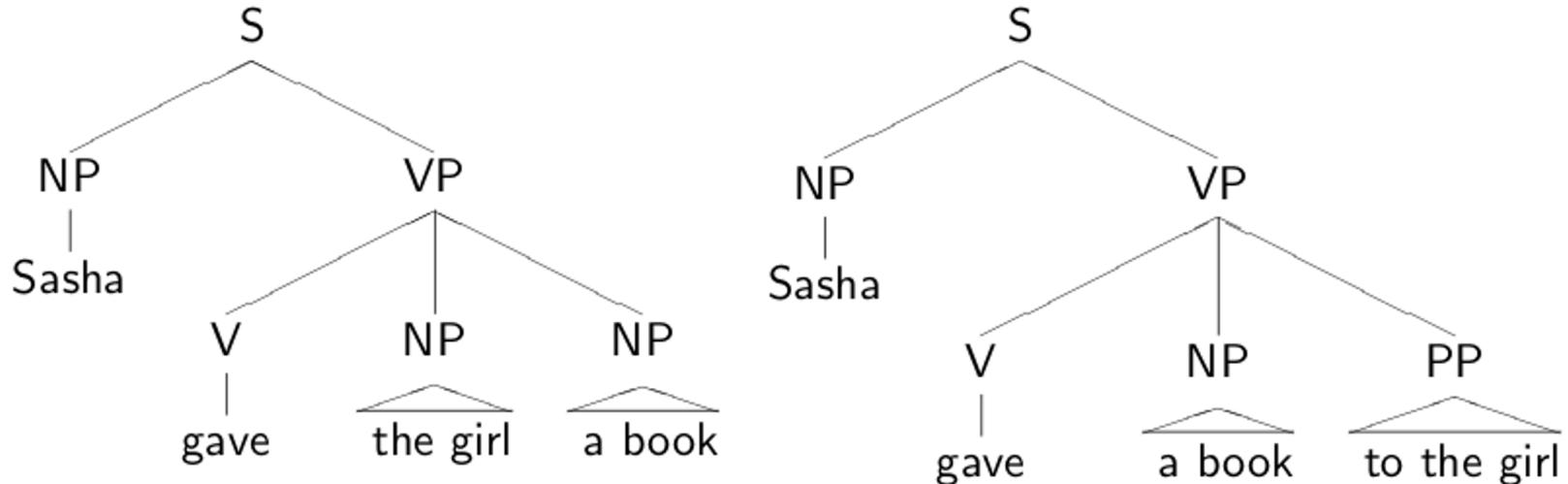
KaiC <-nsubj interactS prep_with-> Sasa

KaiC <-nsubj interactS prep_with-> Sasa conj_and-> KaiA

KaiC <-nsubj interactS prep_with-> Sasa conj_and-> KaiB

[Erkan et al. EMNLP 07, Fundel et al. 2007]

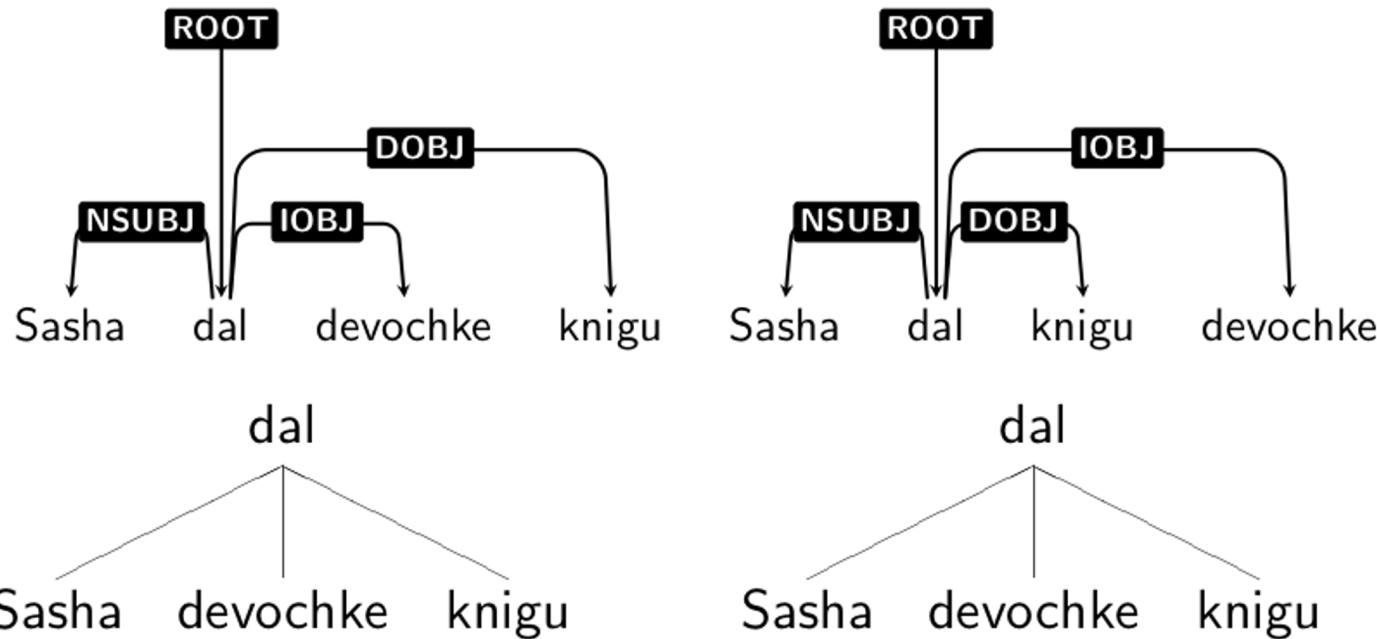
Why dependencies?



Free word-order

- Russian uses morphology to mark relations between words:
 - knigu means book (kniga) as a direct object.
 - devochke means girl (devochka) as indirect object (to the girl).
- So we can have the same word orders as English:
 - *Sasha dal devochke knigu*
 - *Sasha dal knigu devochke*
 - *Sasha devochke dal knigu*
 - *Devochke dal Sasha knigu*
 - *Knigu dal Sasha devochke*

Russian Trees



Pro vs. Cons

pro

- Sensible framework for free word order languages.
- Identifies syntactic relations directly. (using CFG, how would you identify the subject of a sentence?)
- Dependency pairs/chains can make good features in classifiers, for information extraction, etc.
- Parsers can be very fast

con

- The assumption of asymmetric binary relations isn't always right -> **how to parse dogs and cats?**

Outline

- Sintassi a dipendenze
- Parsing a dipendenze: approcci
- Parsing deterministico a “transizioni”
- Parsing a regole per ~~dipendenze a vincoli~~

Tecniche per il dependency parsing

1 • Dynamic programming

2 • Graph algorithms

(considera l'albero come un particolare tipo di grafo
può combinarsi al dynamic programming)

3 • Constituency parsing + conversion

con CKY

ⁿ
in dipendenze

4 • Deterministic parsing

5 • Constraint Satisfaction

"Alessandro Core"

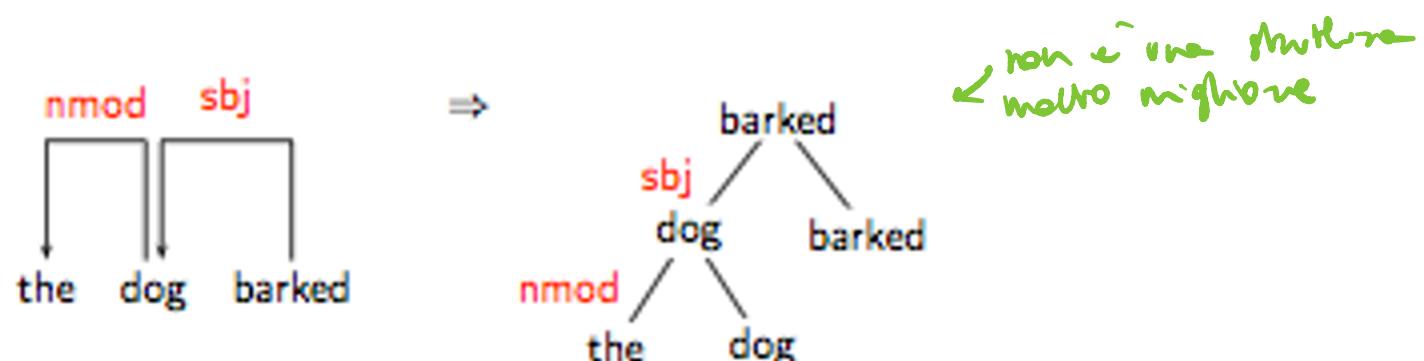


Core
sub/
Alessandro

Tecniche per il dependency parsing

①

- Dynamic programming (simile a CKY)
- Algoritmo simile al parsing delle PCFG lessicalizzate: complessità $O(n^5)$
- Eisner (1996) ha “scoperto” un algoritmo migliore che riduce la complessità a $O(n^3)$



Tecniche per il dependency parsing

i

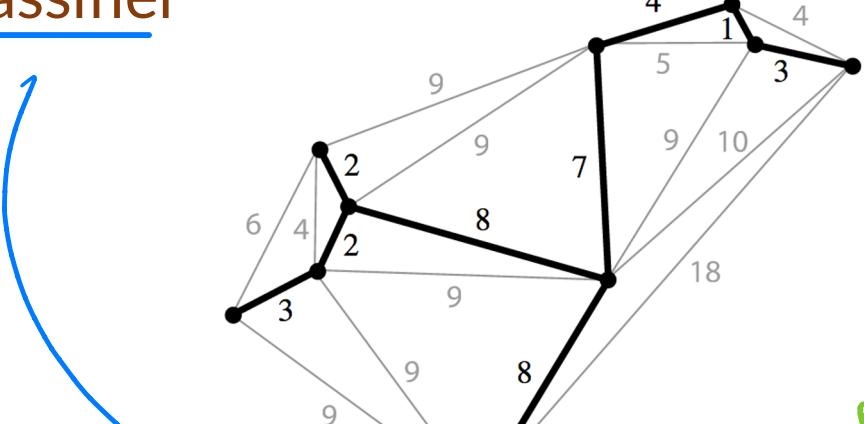
- Graph algorithms (ancora dynamic)

- Si crea un Minimum-Spanning Tree per la frase

- McDonald et al.'s (2005): MSTParser valuta le

- dipendenze indipendentemente usando un ML

- classifier



Parole come "Francesca"

creo 3 nodi e le possibili facce (x ora non ripete) ← ogni faccia ha un peso

vengono da un training sul corpus

potranno poi essere fatte da classificatori

bischi su feature linguistiche che vengono coniate all'interno
del corpus (cf. Hill)

Non è ^{che} conto quante volte c'è "Parlo ora", ma magari
nel corpus c'è un verbo indicativo per. + Parlo,
oppure verbo indicativo per. + nome proprio

l'evento che conta è quello [↓] che identifica l'istante specifico
che ho in inf

Tecniche per il dependency parsing

③ • Constituency parsing + conversion

- Parsing con una grammatica a costituenti e convertito in un formato a dipendenze attraverso delle "tabelle di percolazioni" (teoria X-bar)

4

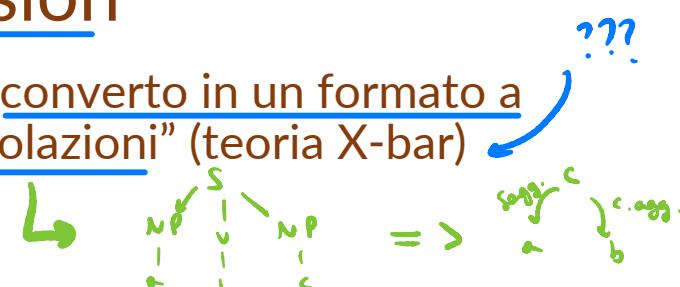
• Deterministic parsing

- Scelte greedy per la creazione di dipendenze tra parole, guidate da machine learning classifiers
- MaltParser (Nivre et al. 2008) -> Transition Based

5

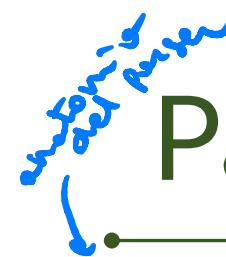
• Constraint Satisfaction

- Vengono eliminate tutte le possibili dipendenze che non soddisfano certi vincoli (hard). (Karlsson 1990), (Lesmo & Torasso 1985): FIDO -> TUP , etc.



Outline

- Sintassi a dipendenze
- Parsing a dipendenze: approcci
- Parsing deterministico a “transizioni”
- Parsing a regole per dipendenze a vincoli



Parser E - MALT

(1) Grammar

Dependency grammar (Automa from TB), ...

! di una competenza nascosta nel classificatore (vs. CKY)

la conoscenza grammatica è dichiarativa; gramm. generativa di Chomsky

(2) Algorithm

I. Search strategy

top-down, ~bottom-up, left-to-right, ...

II. Memory organization

depth-first, back-tracking, dynamic

programming, ...

sente

(3) Oracle

Probabilistic, rule-based, ...

un classificatore del
machine learning

perché non sbagliiamo mai, restituisce
sempre un albero, con una prob.
non nulla (vs CKY)

si fa per "Alessandro corre"
che per "Alessandro corrone"

! il CKY no

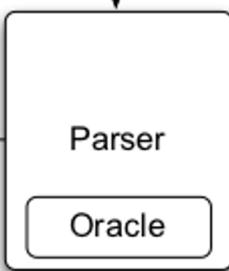
MALT

una sorta di arvore

liste

hashes + leggere

Input buffer



Dependency
Relations

memorie
auxiliarie

Stack



Parser
Oracle

mette in un set

a lbero

se effettivam. come
albero, che come
insieme di coppie
(o triple, se tempo anche
il tipo)

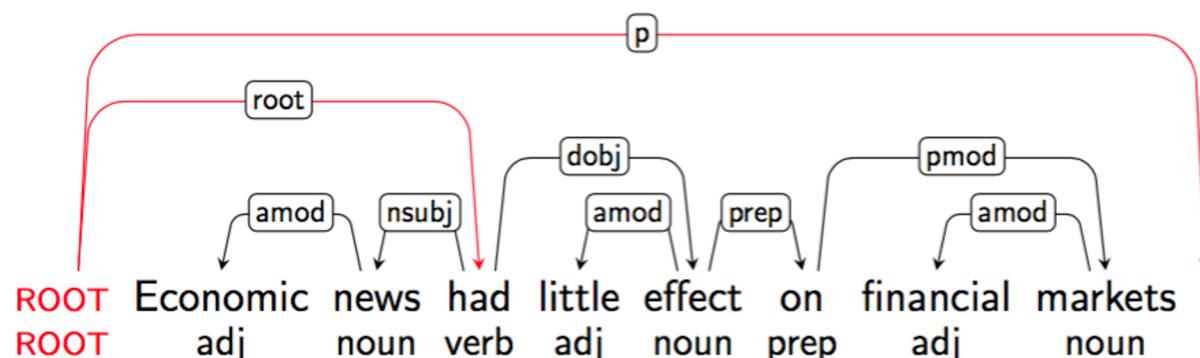
$P \begin{array}{c} / \\ a \\ \backslash \\ F \end{array} \Rightarrow \{(P, a, \text{agg}), (F, a, c.\text{agg}), \dots\}$

facciamo un'ipotesi, di PROGETTIVITÀ

Projectivity

Definition: G is projective

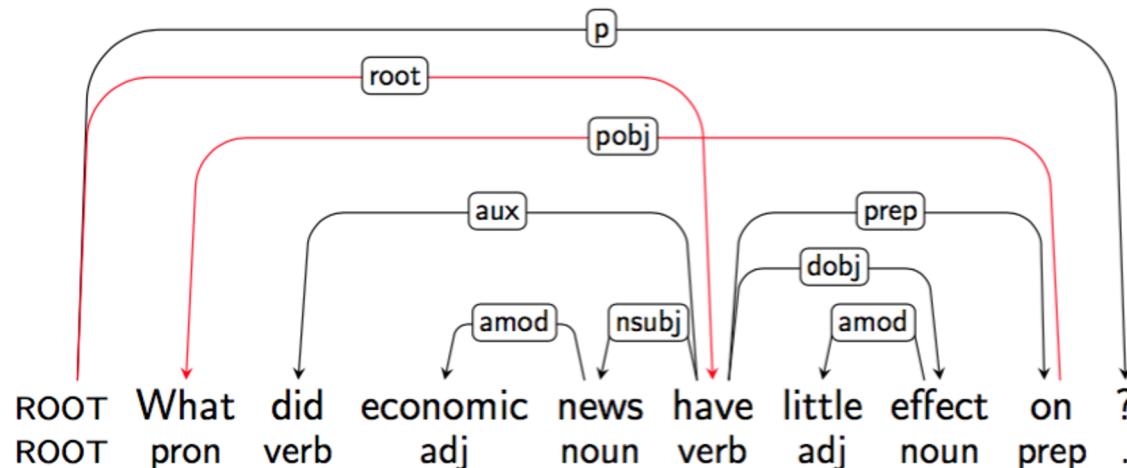
IF $i \rightarrow j$ THEN $i \rightarrow^* k$ for any k such that $i < k < j$ or $j < k < i$



NON Projectivity

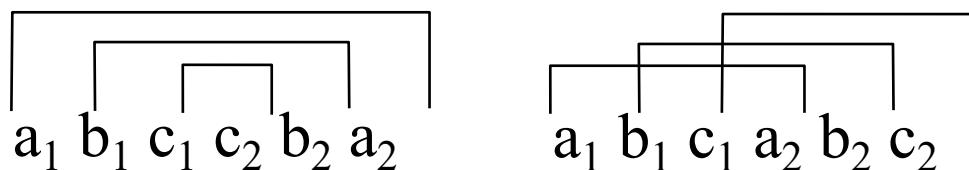
Definition: G is projective

IF $i \rightarrow j$ THEN $i \rightarrow^* k$ for any k such that $i < k < j$ or $j < k < i$



NON Projectivity

- Most theoretical frameworks **do not** assume projectivity
- Non-projective structures are needed to account for long-distance dependencies (e.g. questions) and free word order
- MCSL -> cross-serial !



Transition-Based Parsing

- Ricerca in uno spazio di stati: similitudine con shift-reduce parsing
↳??? (vedere)
- Uno stato è composto da 3 cose:
 - Una stack che contiene le parole parzialmente analizzate
 - Una lista contenente le rimanti parole da analizzare
 - Un insieme contenente le dipendenze create fino a quel punto nell'analisi

l'idea : l'avanzamento nel parsing è come se si muovesse da uno stato a un altro

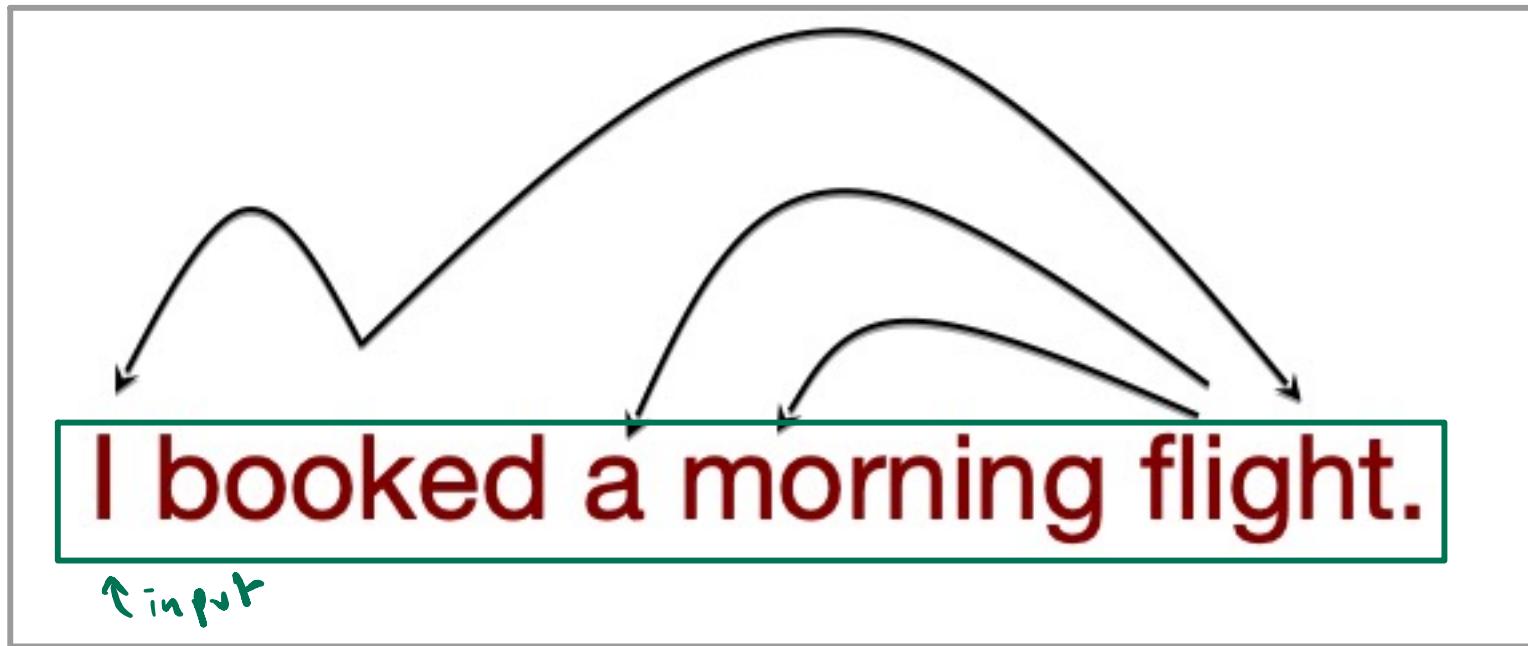
Stati

○ Stato iniziale

- [[root], [sentence], ()]

- [[root], [] (R)]

• Dove R è l'insieme delle dipendenze costruite. [] è la lista vuota poiché tutte le parole sono state analizzate



Esempio

- **Stato iniziale**

[[root], [I booked a morning flight], ()]

- **Stato finale**

[[root], [], ((booked, I) (booked, flight) (flight, a) (flight, morning))]

Operatori di parsing

- Come arrivare dallo stato iniziale allo stato finale?

- Tre operatori di transizione tra stati:

1• Shift ← sposta senza aggiungere nulla

2• Left ← aggiunge una sfida. e finisce

3• Right ← " " " " o destra

le scuglie
l'oracolo

Con questi, che
una seq. di parole,
può creare un
qualsiasi albero
proiettivo

① Operatore Shift

Shift fa 1 cosa:

- prende la prossima parola e la pusha sullo stack
rimuovendola dalla lista

[[root], [I booked a morning flight], ()]

->

[[root, I], [booked a morning flight], ()]

[↑]
top dello stack

② Operatore Left

Left fa 2 cose:

- crea una dipendenza (a,b) tra la prossima parola della lista (a) e la parola sul top dello stack (b)
- Poppa lo stack

[[root, I], [booked a morning flight], ()]

->

[[root], [booked a morning flight], (booked,I)]

↑
rimane sulla lista!

↑
è il dipendente

③ Operatore Right

Right fa 3 cose:

- crea una dipendenza (b,a) tra la prossima parola della lista (a) e la parola sul top dello stack (b)
- rimuove a dalla lista
- Poppa lo stack e mette la parola poppata all'inizio della lista

[[root, booked], [flight], []]

->

[root], [booked], [(booked, flight)]

↑ c'è la ferita

Example

	{[root], [I booked a morning flight], []}

Example

qui dicono < mettendo di sopra quale sia la
seguente corretta

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}
Shift	{[root, booked, a], [morning flight], [(booked, I)]}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}
Shift	{[root, booked, a], [morning flight], [(booked, I)]}
Shift	{[root, booked, a, morning], [flight], [(booked, I)]}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}
Shift	{[root, booked, a], [morning flight], [(booked, I)]}
Shift	{[root, booked, a, morning], [flight], [(booked, I)]}
Left	{[root, booked, a], [flight], [(booked, I), (flight, morning)]}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}
Shift	{[root, booked, a], [morning flight], [(booked, I)]}
Shift	{[root, booked, a, morning], [flight], [(booked, I)]}
Left	{[root, booked, a], [flight], [(booked, I), (flight, morning)]}
Left	{[root, booked], [flight], [(booked, I), (flight, morning), (flight, a)]}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}
Shift	{[root, booked, a], [morning flight], [(booked, I)]}
Shift	{[root, booked, a, morning], [flight], [(booked, I)]}
Left	{[root, booked, a], [flight], [(booked, I), (flight, morning)]}
Left	{[root, booked], [flight], [(booked, I), (flight, morning), (flight, a)]}
Right	{[root], [booked], [(booked, I), (flight, morning), (flight, a), (booked, flight)] }

Example

	$\{[\text{root}], [\text{I booked a morning flight}], []\}$
Shift	$\{[\text{root, I}], [\text{booked a morning flight}], []\}$
Left	$\{[\text{root}], [\text{booked a morning flight}], [(\text{booked, I})]\}$
Shift	$\{[\text{root, booked}]\}, [\text{a morning flight}], [(\text{booked, I})]\}$
Shift	$\{[\text{root, booked, a}], [\text{morning flight}], [(\text{booked, I})]\}$
Shift	$\{[\text{root, booked, a, morning}], [\text{flight}], [(\text{booked, I})]\}$
Left	$\{[\text{root, booked, a}], [\text{flight}], [(\text{booked, I}), (\text{flight, morning})]\}$
Left	$\{[\text{root, booked}], [\text{flight}], [(\text{booked, I}), (\text{flight, morning}), (\text{flight, a})]\}$
Right	$\{[\text{root}], [\text{booked}], [(\text{booked, I}), (\text{flight, morning}), (\text{flight, a}), (\text{booked, flight})] \}$
Right	$\{[], [\text{root}], [(\text{booked, I}), (\text{flight, morning}), (\text{flight, a}), (\text{booked, flight}), (\text{root, booked})] \}$

dispens.[↑] fitting = dire
 che booked è la root-ve

Example

un albero di n parole, se considero anche la dipend. con root, ho n dipendenze binarie

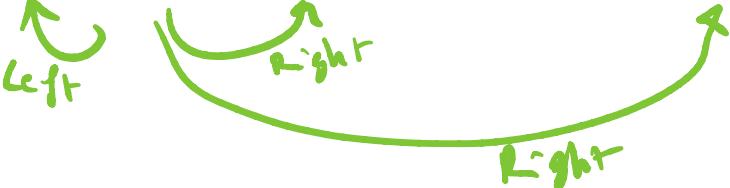
	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left	{[root], [booked a morning flight], [(booked, I)]}
Shift	{[root, booked]}, [a morning flight], [(booked, I)]}
Shift	{[root, booked, a], [morning flight], [(booked, I)]}
Shift	{[root, booked, a, morning], [flight], [(booked, I)]}
Left	{[root, booked, a], [flight], [(booked, I), (flight, morning)]}
Left	{[root, booked], [flight], [(booked, I), (flight, morning), (flight, a)]}
Right	{[root], [booked], [(booked, I), (flight, morning), (flight, a), (booked, flight)] }
Right	{[], [root], [(booked, I), (flight, morning), (flight, a), (booked, flight), (root, booked)] }
Shift	{[root], [], [(booked, I), (flight, morning), (flight, a), (booked, flight), (root, booked)] }

Parsing search algorithm

```
depParse(s)
    state = initialState(s)
    while state is not final
        op = oracle(state)
        state = apply op to state
    return relations.state()
```

↑
è molto semplice, la
complessità è spesso molto bassa

Bisogna provare provare provare

- *Paolo ama Francesca dolcemente*
- ... 

2 problemi

1• Operatori e dipendenze -> relazioni “labelled”:

subject, direct object, indirect object, etc.

2• Quale operatore (L, R, S) usare ad ogni passo?

-> Oracolo e memory

↳ è statisticico . Come calcolo le statistiche ? uso un TB

① Problema 1: Operatori e dipendenze

come etichetto le frecce?

costruisco una
verticale biparte
di ogni operatore

- n dipendenze $\rightarrow 2n+1$ operatori
- $\{Left, Right\} \times \{\text{tutte le relazioni}\} + Shift$
- Es: Left_subj, Right_subj, Left_obj, Right_obj, Shift

↓
*con più lo prob-encore + base
x ogni operat. → + spaziosità dei dati*

Example

	{[root], [I booked a morning flight], []}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}

Example

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left_Subj	{[root], [booked a morning flight], [(subj, booked, I)]}



Problema 2: Oracolo a regole

- Scelgo l'operatore da applicare  sando un insieme di regole sul valore delle features dello stato corrente
- Ad esempio: **IF** la parola al top dello stack  è un pronome e la parte rimanente dello stack è solo “root” e la prima parola della lista è un verbo **THEN** OP=Left_Subj

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left_Subj	{[root], [booked a morning flight], [(subj, booked, I)]}

Problema 2: Oracolo probabilistico (MALT)

- Uso un sistema di ML per addestrare un classifier per scegliere l'operatore -> oracolo=classifier
- Il classifier sarà basato sulle feature dello stato: $CL(\{[root, I], [booked a morning flight], []\}) \rightarrow OP=Left_Subj$

	{[root], [I booked a morning flight], []}
Shift	{[root, I], [booked a morning flight], []}
Left_Subj	{[root], [booked a morning flight], [(subj, booked, I)]}

3 punti importanti

L'uso del ML comporta:

1. Scoprire le features linguisticamente significative

* per trovare, con queste features, quante volte ho fatto S,L,R all'interno del TG

2. Costruire il training data -> treebanks

3. Training algorithm

così invece che dare all'oscuro
TUTTI i dettagli su uno stato (+ farsi
dare l'operat. da fare), gli dò solo
quegli linguisticamente significativi.

! due stati che hanno le stesse
features sono praticamente lo stesso stato

senza → significa che operat. fare con una parola sconosciuta non so che fare

Punto 1: features template

Tipiche features sono/riguardano:

- posizioni nello stato
 - stack
 - list (buffer)
 - albero parziale
- attributi di alcune parole
 - PoS del top dello stack
 - PoS della terza parola nella lista
 - Lemma del top della lista
 - Head del top della lista
 - dipendenze del top della lista
 - etc ...

il # degli stati è infinito
le prob. dello stato attuale potrebbe
essere 0 (xe non lo si è mai
visto nel tb)

```
depParse(s)
state = initialState(s)
while state is not final
    op = oracle(state)
    state = apply op to state
return relations.state()
```

concretamente quando fai features engineering
è come dire

valore delle features

Punto 1: features template

Tipiche features sono/riguardano:

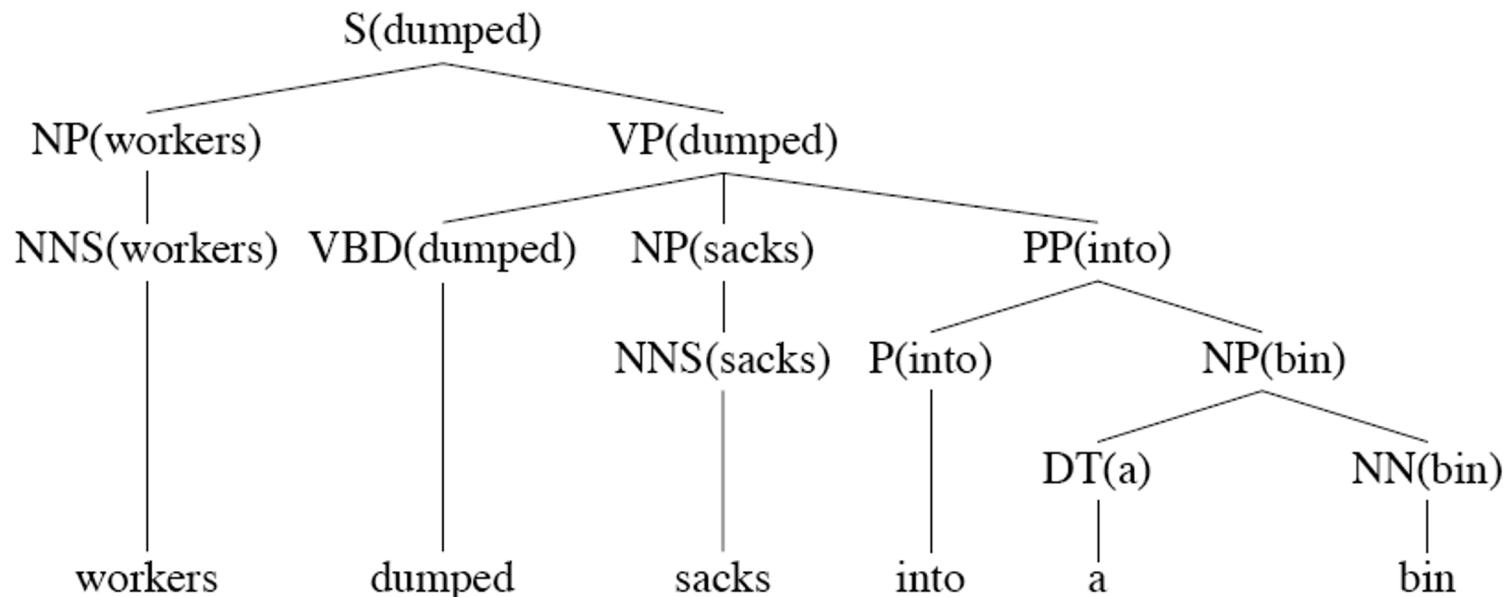
- posizioni nello stato
 - stack
 - list (buffer)
 - albero parziale
- attributi di alcune parole
 - PoS del top dello stack
 - PoS della terza parola nella lista
 - Lemma del top della lista
 - Head del top della lista
 - dipendenze del top della lista
 - etc ...

stack, buffer, word, tag

- $s_1.w$
- $s_2.w$
- $s_1.t$
- $s_2.t$
- $b_1.w$
- $b_1.t$
- $s_1.wt$

Punto 2: training data

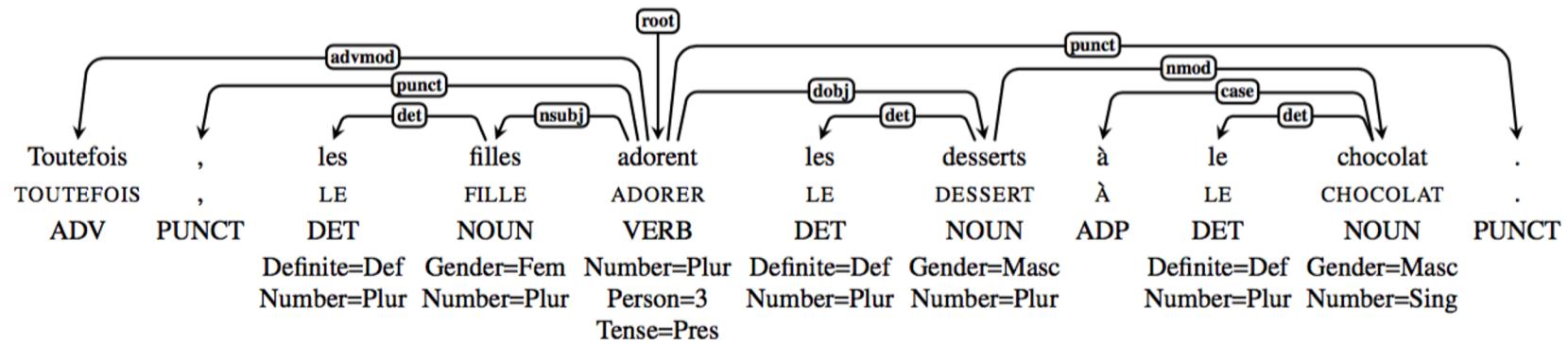
- TB a dipendenze
- TB a costituenti + conversione (percolazione)



Universal Dependency TB

- 100 TBs , 70 languages
 - <https://universaldependencies.github.io/docs/>
- No-white space tokenization (cf. TUT later)
- Morphology: Lemma+PoS+FeatureSet
- 3 syntactic principles (40 **basic** relations)
 - content words are related by dependency relations
 - function words attach to the content word they further specify
 - punctuation attaches to the head of the phrase or clause in which it appear

Universal Dependency TB



Universal Dependency TB

```
# isst_tanl sent_id 5
```

1	Inconsueto	inconsueto	ADJ	A	Gender=Masc Number=Sing	2	amod	-	-
2	allarme	allarme	NOUN	S	Gender=Masc Number=Sing	0	root	-	-
3-4	alla	-	-	-	-	-	-	-	-
3	a	a	ADP	E	-	5	case	-	-
4	la	il	DET	RD	Definite=Def Gender=Fem Number=Sing PronType=Art	5	det	-	-
5	Tate	Tate	PROPN	SP	-	2	nmod	-	-
6	Gallery	Gallery	PROPN	SP	-	5	name	-	-
7	:	:	PUNCT	FC	-	2	punct	-	-

Punto 3: training algorithm

- Dato un albero del TB, devo ricostruire tutte le scelte giuste del mio parser per costruire quello specifico albero
- Ricostruendo ogni passo del parsing:
 - Left se ottengo la dipendenza che correttamente si trova nell'albero
 - Right se ottengo la dipendenza che correttamente si trova nell'albero AND tutte le altre dipendenze associate alla parola figlia si trovano già nell'albero
 - Altrimenti Shift

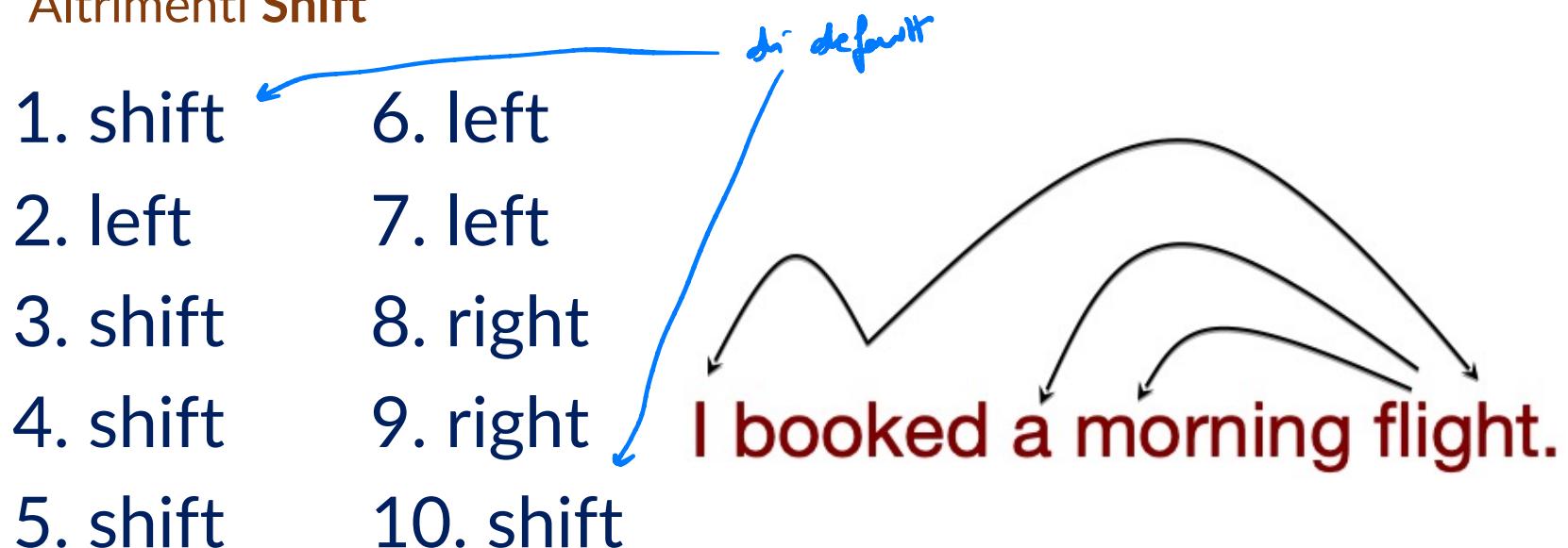
C'è più di una dipendenza e non sono tutte nella lista



=> Sequenza di S, L, R

Reverse Engineering

- **Left** se ottengo la dipendenza che correttamente si trova nell'albero
- **Right** se ottengo la dipendenza che correttamente si trova nell'albero
AND se tutte le altre dipendenze associate alla parola figlia si trovano già nell'albero
- Altrimenti **Shift**



features -> operatore da scegliere

Tipiche features sono/riguardano:

- *posizioni* nello stato
 - stack
 - list (buffer)
 - albero parziale
- attributi di alcune parole
 - PoS del top dello stack
 - PoS della terza parola nella lista
 - Lemma del top della lista
 - Head del top della lista
 - dipendenze del top della lista
 - etc ...

stack, buffer, word, tag, operator

- $\langle s_1 .w, op \rangle$
- $\langle s_2 .w, op \rangle$
- $\langle s_1 .t , op \rangle$
- $\langle s_2 .t , op \rangle$
- $\langle b_1 .w, op \rangle$
- $\langle b_1 .t , op \rangle$
- $\langle s_1 .wt , op \rangle$

United canceled the morning flight to Huston

- Stack:
[root,canceled,flights]
- Word buffer:
- [to Houston]
- Relations :
(canceled→United)
- (flights → morning)
- (flights → the)

United canceled the morning flight to Huston

- Stack:
[root,canceled,flights] $\langle s_1.w = \text{flights}, op = \text{shift} \rangle$
 $\langle s_2.w = \text{canceled}, op = \text{shift} \rangle$
- Word buffer:
[to Houston] $\langle s_1.t = \text{NNS}, op = \text{shift} \rangle$
 $\langle s_2.t = \text{VBD}, op = \text{shift} \rangle$
 $\langle b_1.w = \text{to}, op = \text{shift} \rangle$
 $\langle b_1.t = \text{TO}, op = \text{shift} \rangle$
- Relations :
(canceled → United) $\langle s_1.wt = \text{flightsNNS}, op = \text{shift} \rangle$
(flights → morning)
(flights → the)

Scoring function: $c=$ configuration $t=$ transition

- Scoring function: $S(c,t) = w \cdot f(c,t)$ dove:
 - $f(c,t)$ = vettore di feature sulla configurazione c e la transizione t
 - w = vettore di pesi [w_i = peso della feature $f_i(c, t)$]
- Simple classification problem:
 - Apprendi i pesi che massimizzano lo score della transizione corretta per tutte le configurazioni nel training set
 - Le configurazioni sono la derivazione parziali corrette dell'albero a dipendenze
- L'apprendimento è locale alla configurazione usando però delle feature globali

Pseudo-projective parsing [Nivre and Nilsson 2005]

- Preprocess training data, post-process parser output
- Approximate encoding with incomplete coverage
- Relatively high precision but low recall

Outline

- Sintassi a dipendenze
- Parsing a dipendenze: approcci
- Parsing deterministico a “transizioni”
- Parsing a regole per dipendenze a vincoli

Parser F - TUP

(1) Grammar

Dependency grammar (constraints), ...

(2) Algorithm

I. Search strategy

top-down, ~bottom-up, left-to-right, ...

II. Memory organization

depth-first, back-tracking, dynamic
programming, ...

(3) Oracle

Probabilistic, rule-based, ...

Turin University Parser

- Un parser a **dipendenze** bottom-up di ampia copertura basato su regole.
- **Regole** per: *Chunking*, *Coordination*, *Verb-SubCat*
- Dipendenze
 - Morpho-syntactic
 - Syntactic-functional
 - Semantic

Turin University Parser

Chuncking nominale (non verbale)

(ADJ-QUALIF

BEFORE (ADV (TYPE MANNER))

ADVMOD-MANNER)

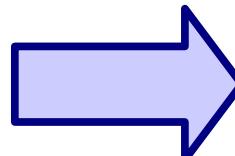
IF un avverbio di tipo **MANNER** (modo) precede

immediatamente un aggettivo qualificativo **THEN** esso

dipende da esso attraverso una dipendenza **ADVMOD-**

MANNER

... davvero veloce ...



veloce

↓ ADVMOD-MANNER

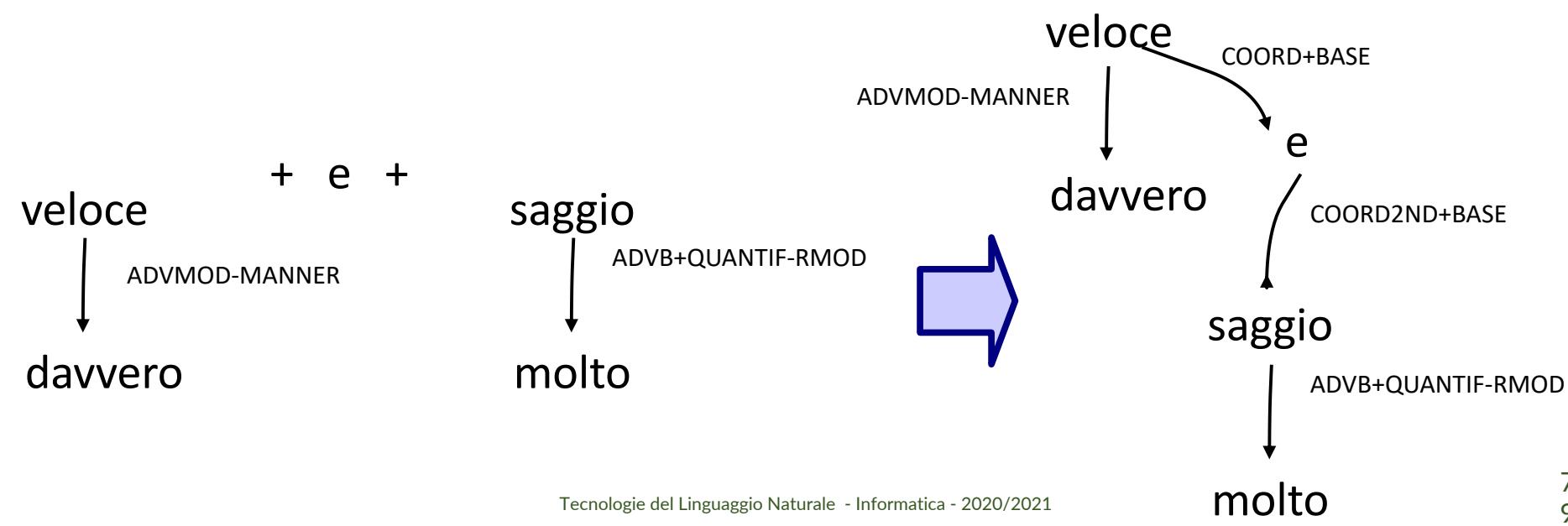
davvero

Turin University Parser

Coordination:

rimuovo l'ambiguità delle congiunzioni con delle regole

... davvero veloce e molto saggio ...



Turin University Parser

Verb-SubCat: regole verbali basate su una tassonomia di classi per la sottocategorizzazione

VERBS

TRANS

...

INTRANS

...

INTRANS-INDOBJ-PRED (Ex. "La casa gli
sembra bella")

...

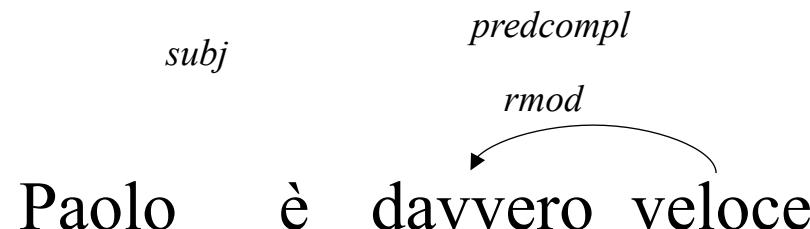
Turin University Parser

Paolo è davvero veloce

Chunking



Verb-SubCat



Turin University Parser

Oggi ultimo giorno del mese di Giugno,

con valori di temperatura superiori alla media



rmod

separator

rmod

rmod

rmod

rmod

arg

rmod arg

rmod arg

arg

rmod

arg

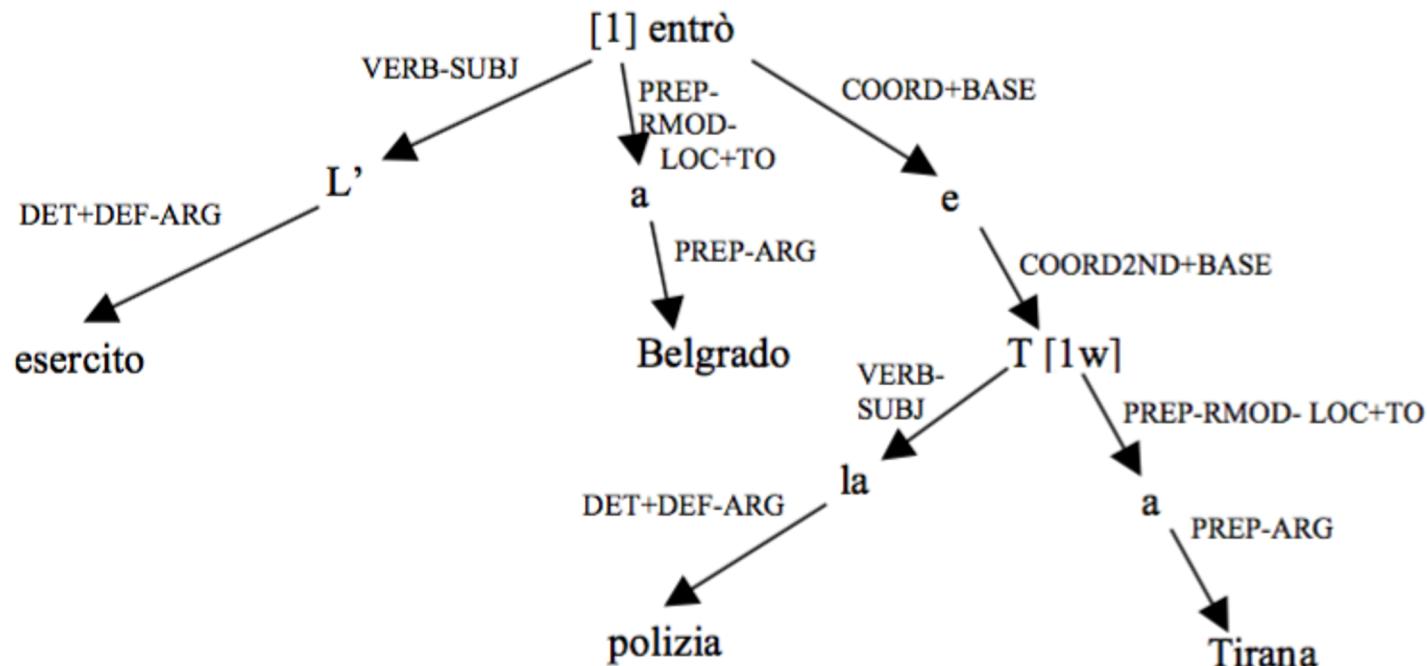
arg

arg

Oggi ultimo giorno del mese di giugno , con valori di temperatura superiori alla media

Turin University Treebank

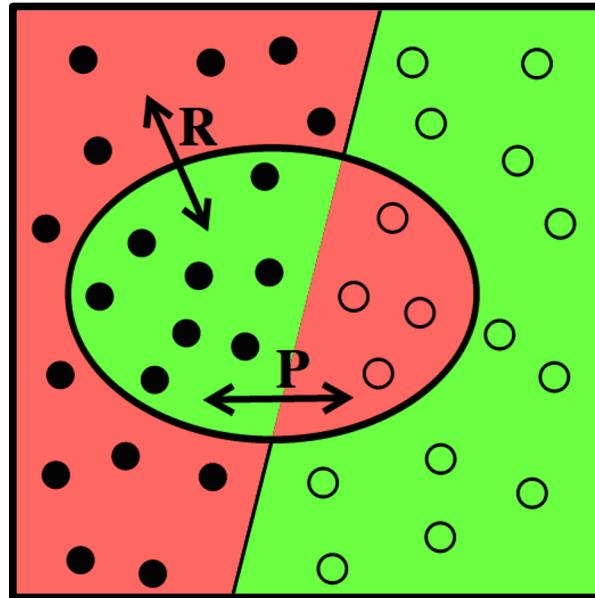
<http://www.di.unito.it/~tutreeb/>



Valutazione

Non c'è Precision e Recall, perché basta il solo numero:
date n parole ho n dipendente → Precision e Recall
coincidono → n parte di Accuracy

- Accuracy
- Precision vs. Recall
- F-score
- Unlabelled vs. Labelled



$$P = \frac{T^\bullet}{T^\bullet + F^\bullet}$$

$$R = \frac{T^\bullet}{T^\bullet + F^\circ}$$

Precision=Recall in dependency parsing

$G = \text{Golden tree}$

$S = \text{Parser tree} \rightarrow |G| = |S|$

$$P = \frac{T^+}{T^+ + F^+} \quad R = \frac{T^+}{T^+ + F^-}$$

$$T^+ = |S \cap G|$$

$$F^+ = |S - G| = |S - (S \cap G)| = |S| - |S \cap G|$$

$$F^- = |G - S| = |G - (G \cap S)| = |G| - |G \cap S|$$