STRING KERNEL

- Given two strings, the number of matches between their substrings is computed
- E.g. *Bank* and *Rank*
  - *B, a, n, k, Ba, Ban, Bank, an, ank, nk*
  - *R, a , n , k, Ra, Ran, Rank, an, ank, nk*
- String kernel over sentences and texts
- Huge space but there are efficient algorithms
  - Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello; Watkins, Chris (2002). "*Text classification using string kernels*". Journal of Machine Learning Research: 419–444.

STRING KERNEL

- A function that give two strings s and *t* is able to compute a real number *k(s,t)* such that
  - two vectors exist $\vec{s}$ and $\vec{t}$
  - $\vec{s}$ and $\vec{t}$ are unique for s and *t*
  - (the vectors represents strings by ***embedding*** their crucial properties!!)
  - k(s,t) = $\vec{s} \times \vec{t}$
- We will see how vectors $\vec{s}$ and $\vec{t}$ are defined in $\mathbb{R}^{\infty}$, as the numer of strings of arbitrary length over an alphabet is infinite
- IDEA: Define a space whereas each substring is a dimension

KERNEL TRA *BANK* E *RANK*

B, a, n, k, Ba, Ban, Bank, an, ank, nk, Bn, Bnk, Bk and ak are the substrings of $Bank$.

R, a, n, k, Ra, Ran, Rank, an, ank, nk, Rn, Rnk, Rk and ak are the substrings of $Rank$.

ϕ

ϕ(Bank)= (λ , 0, λ , λ , λ , λ^2 , λ^2, λ^3 , 0 , λ^4 , 0 , λ^2 , λ^3 , λ^3 , ...

ϕ(Rank)= (0 , λ, λ , λ , λ , 0 , 0 , 0 , λ^3 , 0 , λ^4 , λ^2, λ^3 , λ^3 , ...

B , R, a, n , k, Ba, Ra, Ban, Ran, Bank, Rank, an, ank , ak ...

- Common substrings:
  - *a*, *n*, *k*, *an*, *ank*, *nk*, *ak*
- Notice how these are the same subsequences as between
  - *Schrianak* and *Rank*

FORMALLY ...

Sottosequenza di indici ordinati e non contigui di (*1, ...* |s|)

$$s = s_1, .., s_{|s|}$$

$$\vec{I} = (i_1, ..., i_{|u|})$$

$$u = s[\vec{I}]$$, substring of s defined by $\overrightarrow{I}$

$$\phi_u(s) = \sum_{\vec{I}:u=s[\vec{I}]} \lambda^{l(\vec{I})}, \text{ con } l(\vec{I}) = i_{|u|} - i_1 + 1$$

$$K(s,t) = \sum_{u\in\Sigma^*} \phi_u(s)\cdot\phi_u(t) = \sum_{u\in\Sigma^*} \sum_{\vec{I}:u=s[\vec{I}]} \lambda^{l(\vec{I})} \sum_{\vec{J}:u=t[\vec{J}]} \lambda^{l(\vec{J})} =$$

$$= \sum_{u\in\Sigma^*} \sum_{\vec{I}:u=s[\vec{I}]} \sum_{\vec{J}:u=t[\vec{J}]} \lambda^{l(\vec{I})+l(\vec{J})}, \text{ con } \Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

AN EXAMPLE OF STRING KERNEL COMPUTATION

- $\phi_{\mathrm{a}}(\mathtt{Bank}) = \phi_{\mathrm{a}}(\mathtt{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(2-2+1)} = \lambda,$
- $\phi_{\mathrm{n}}(\mathtt{Bank}) = \phi_{\mathrm{n}}(\mathtt{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(3-3+1)} = \lambda,$
- $\phi_{\mathrm{k}}(\mathtt{Bank}) = \phi_{\mathrm{k}}(\mathtt{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(4-4+1)} = \lambda,$
- $\phi_{\mathrm{an}}(\mathtt{Bank}) = \phi_{\mathrm{an}}(\mathtt{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(3-2+1)} = \lambda^2,$
- $\phi_{\mathrm{ank}}(\mathtt{Bank}) = \phi_{\mathrm{ank}}(\mathtt{Rank}) = \lambda^{(i_1 - i_3 + 1)} = \lambda^{(4-2+1)} = \lambda^3,$

$\phi_{\mathrm{nk}}(\mathtt{Bank}) = \phi_{\mathrm{nk}}(\mathtt{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4-3+1)} = \lambda^2,$

$\phi_{\mathrm{ak}}(\mathtt{Bank}) = \phi_{\mathrm{ak}}(\mathtt{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4-2+1)} = \lambda^3.$

It follows that $K(\mathtt{Bank}, \mathtt{Rank}) = (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3) \cdot (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3)$
$= 3\lambda^2 + 2\lambda^4 + 2\lambda^6.$

LEARNING UNDER KNOWLEDGE REPRESENTATION CONSTRAINTS

LINGUISTIC KERNELS AND LANGUAGE LEARNING

TREE KERNELS

- String kernels adopt a structured approach to kernel estimation and are very useful in NLP and Web Mining tasks
- However, what has been defined over sequences can be profitably exploited also in the treatment of more complex structures
  - Trees whose parent relationship determine subsequences in terms of
    - Multiple paths from the root to the leaves
    - Ordered sets of children (i.e. sequences of immediately dominated nodes) of every node in the tree
  - Graphs, whose structure can be captured by several trees (subgraphs) and thus characterized by multiple subsequences

TREE KERNELS

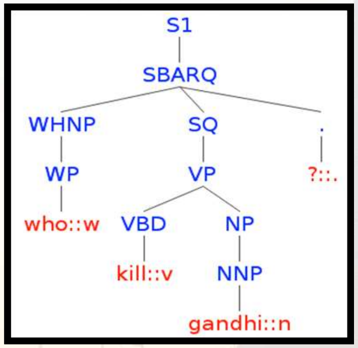
- Applications are related to text processing tasks such as
  - Syntactic parsing, when SVM classification is useful to select the best parse tree among multiple legal grammatical interpretations
  - Question Classification, where SVM classification is applied to the recognition of the target of a question (e.g. a person such as in "*Who is the inventor of the light?*" vs. a place as in "*Where is Taji Mahal?*"

or to pattern recognition (e.g. in bioinformatics the classification of protein structures)

TREE KERNELS

Modeling syntax in Natural Language learning task is complex, e.g.

- Question Classification
- Semantic role relations within predicate argument structures
- Dialogue structures
- Sense Hierarchies



Tree kernels are natural way to exploit syntactic information from sentence parse trees

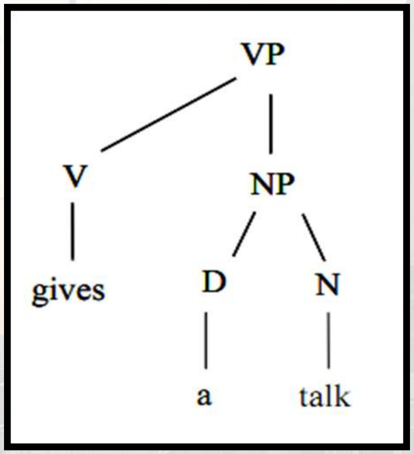
- useful to engineer novel and complex features.

TREE STRUCTURES AND NATURAL LANGUAGE

- PARSING: Breaking down a text into its component parts of speech (according to a formal grammar) with an explanation of the form, function, and syntactic relationship of each part

- INPUT: *gives a talk*

- Output : a *costituency* tree



Chomsky, N. 1957. Syntactic Structures. The Hague/Paris: Mouton.

A DIGRESSION: NL SYNTAX AND SEMANTICS

Lesson 1: March 1°, 2023

SYNTANCTIC PARSING AND CFG

- Formal Definition: a context free grammar (CFG) is a 4-tuple

  $G=(N, \Sigma, R, S)$

  where:

  - N is a set of non-terminal symbols
  - Σ is a set of terminal symbols
  - R is a set of production rules of the form $X \rightarrow Y_1 Y_2 \cdots Y_n$ for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
  - $S \in N$ is a distinguished start symbol

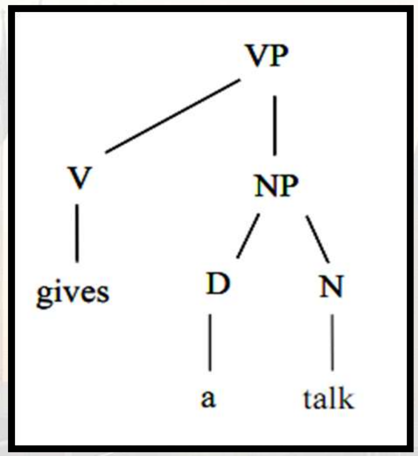
SYNTANCTIC PARSING AND CFG (2)

- N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
- S = S, Σ = {*sleeps, saw, gives, man, woman, telescope, talk, with, in*}

R=

S	→	NP VP
S	→	VP NP
	…	
VP	→	Vi
VP	→	Vt NP
VP	→	VP PP
	…	
NP	→	DT NN
NP	→	NP PP
	…	
PP	→	IN NP
	…	

Vi	→	*sleeps*
Vt	→	*saw*
Vt	→	*gives*
	…	
NN	→	*man*
NN	→	*woman*
NN	→	*telescope*
	…	
DT	→	*the*
DT	→	*a*
	…	
IN	→	*with*
IN	→	*in*



Note
S=sentence
VP=verb phrase
NP=noun phrase
PP=prepositional phr.
DT=determiner
Vi=intransitive verb
Vt=transitive verb
NN=noun
IN=preposition

SYNTAX: PHRASE STRUCTURE GRAMMARS (CHOMSKY, 75)

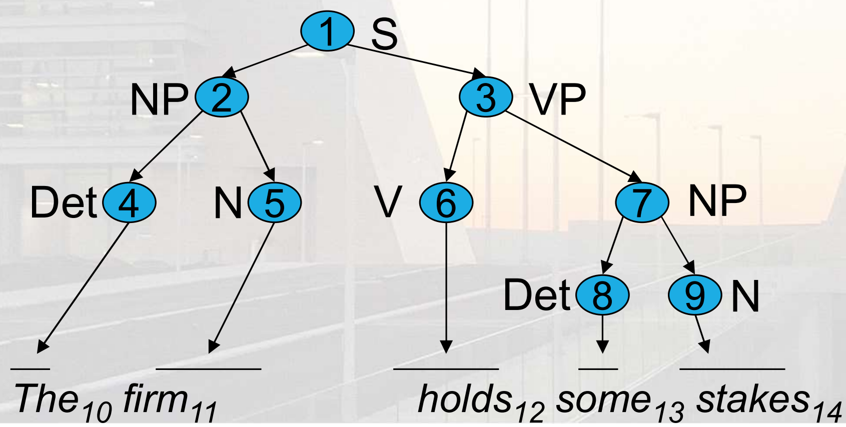
"The firm holds some stakes"

Symbol Vocabulary: Vn={S,NP,VP,Det,N}, Axiom: S

Productions: {S⟶NP VP, VP⟶V NP, NP⟶Det N}

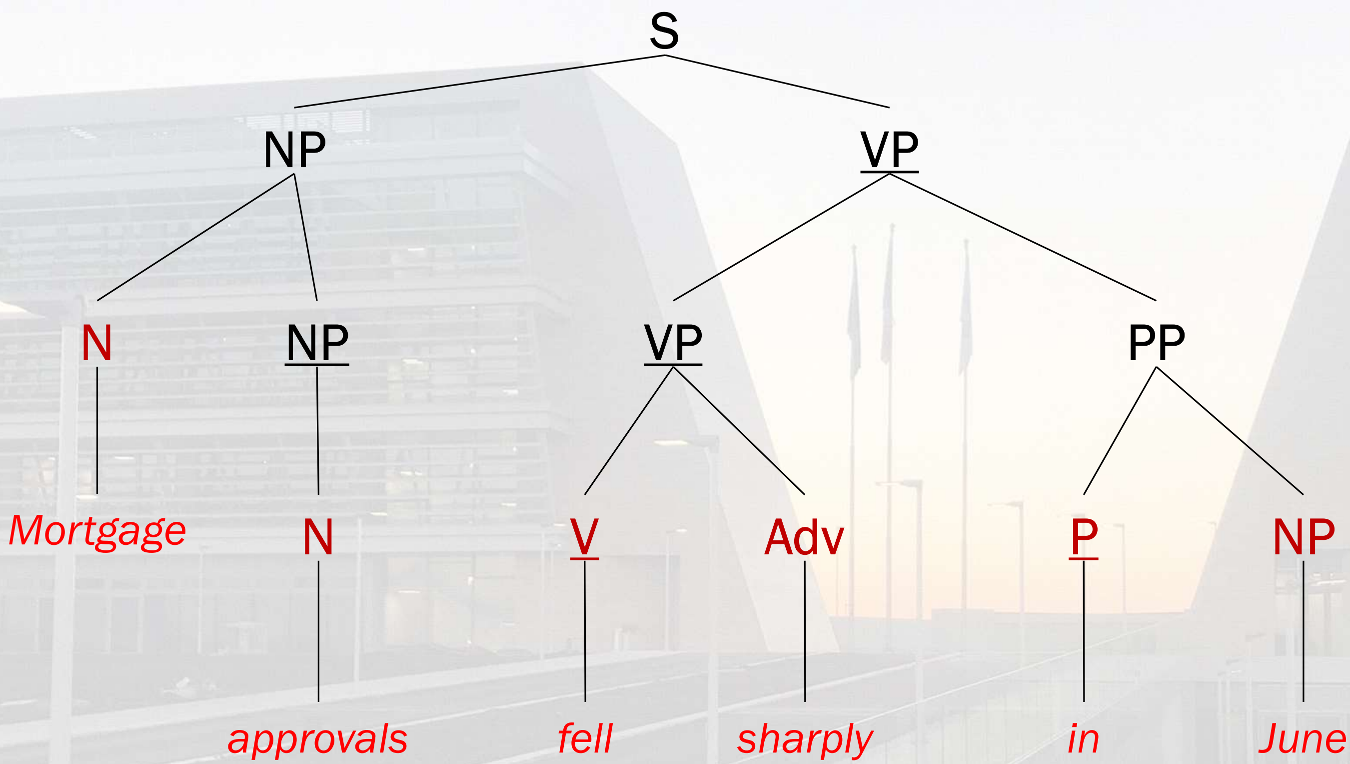
A Derivation is the repreesentation of the cascade of rules used to rewrite S, e.g. :

- S > NP VP > Det N VP > *The* N VP > *The firm* VP > *The firm* V NP > *The firm holds* NP > *The firm holds* Det N > *The firm holds some* N > *The firm holds some stakes*

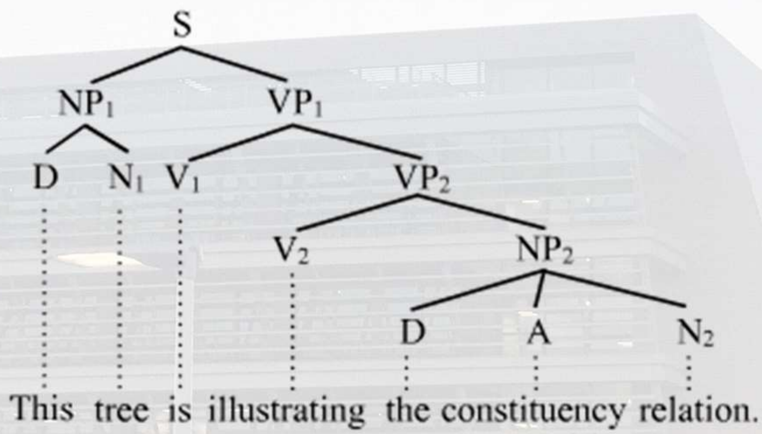


CONSTITUENT-BASED PARSING (WITH MARKED HEADS)

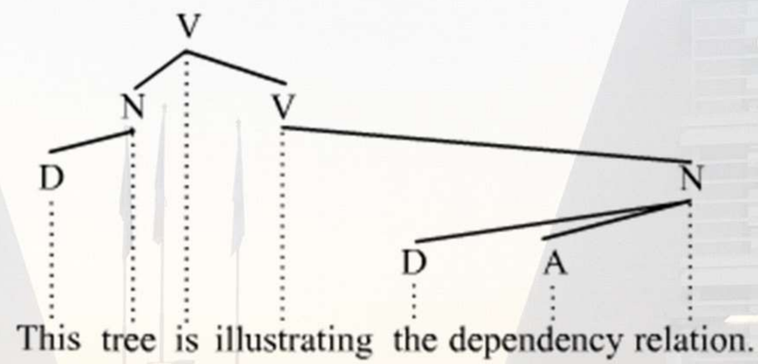
- Marked Heads denote semantic elements of the sentence and facilitate meaning extraction



DIFFERENT GRAMMATICAL THEORIES CORRESPOND TO DIFFERENT TREES: CONSTITUENCY-RELATIONS VS. DEPENDENCY RELATIONS

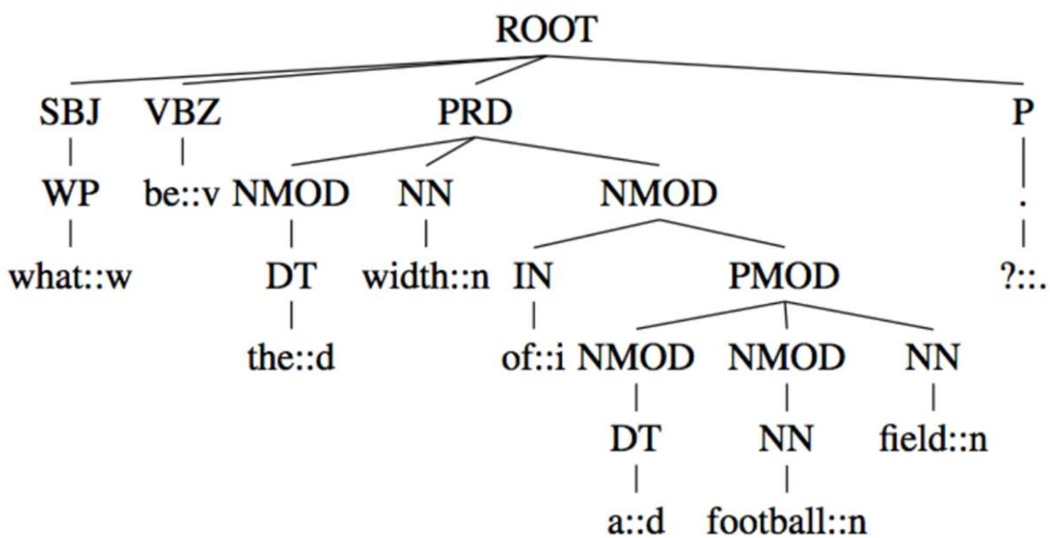


Constituency relation (PSG)



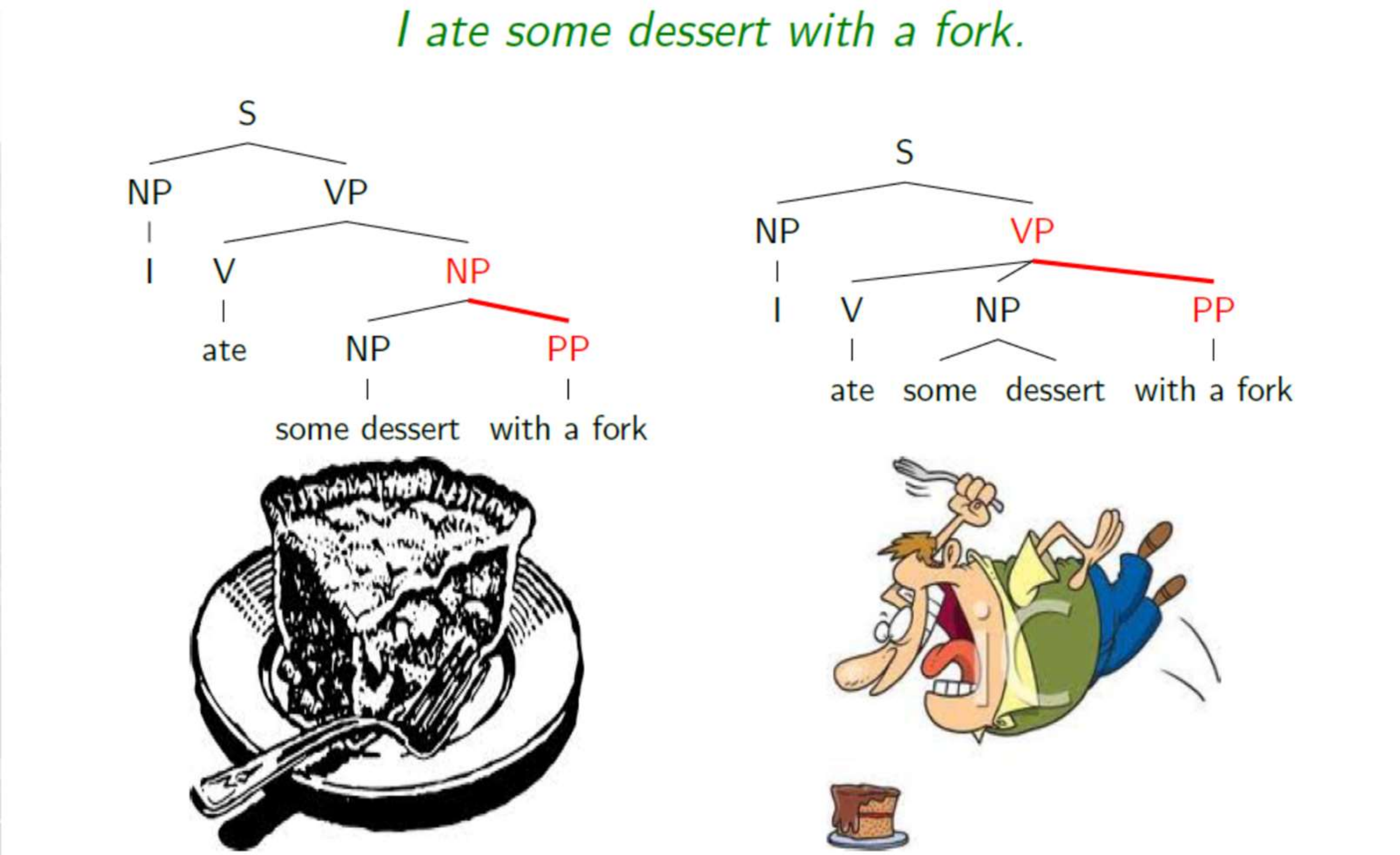
Dependency relation

MARKING GRAMMATICAL NODES FIRST: GRCTs



Grammatical Relation Centered Tree (GRCT)

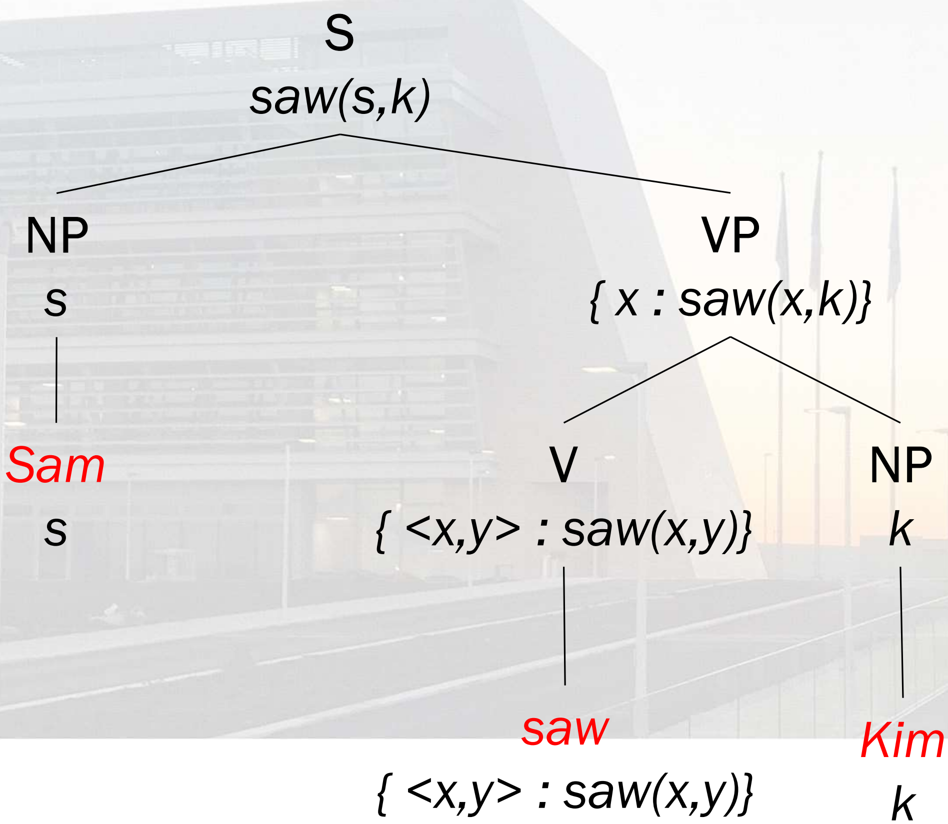
GRAMMARS & AMBIGUITY



A TRUTH-CONDITIONAL PROGRAM FOR NL SEMANTICS

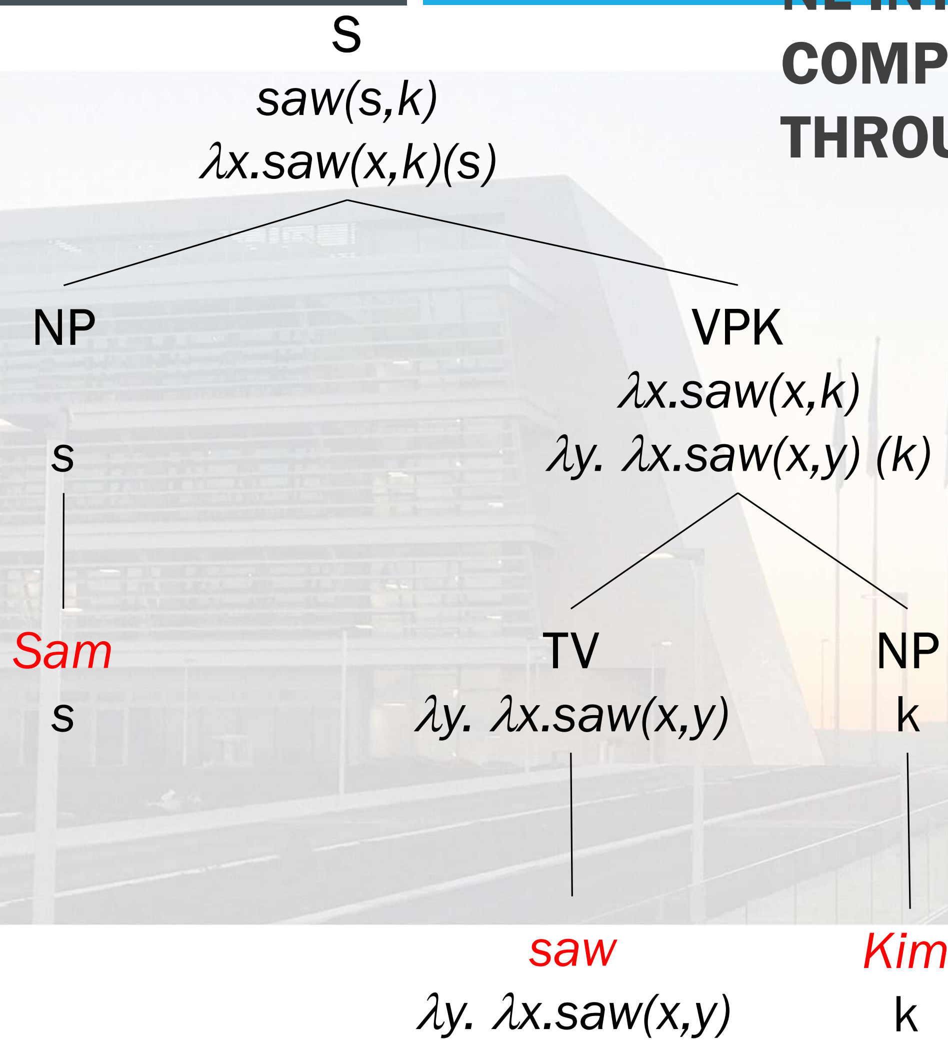
- To define a representation for the semantics of sentences in natural languages correspond to producing:
  - Quantified Logical Forms
  - Relational Forms (ground, data record in DataBases)
  - (Document) Bag-of-Word Vectors (in the style of Rocchio-like models)
- *Two TASKS*
  - ***Interpretation:*** *To determine a procedure for (automatically) generating such a (selected) representation*
  - ***Decision-making from textual data:*** *To (formally) support the different inferences based on the representation that are harmonic with the ones caried out by speakers and hearers of the language*
    - *Automatic Theorem Proving (NL Inference, Paraphrasing, Entity Extraction, Summarization)*
    - *Automatic compilation of SQL queries from natural language questions (Text-to-SQL task)*
    - *CI fanno addestrare i classificatori per categorizzare i testi*

A TRUTH CONDITIONAL SEMANTICS



Sam saw Kim

NL INTERPRETATION AS COMPOSITIONAL PROCESSING THROUGH *LAMBDA ESPRESSIONS*



Sam watched Kim

Sam was screeing Kim

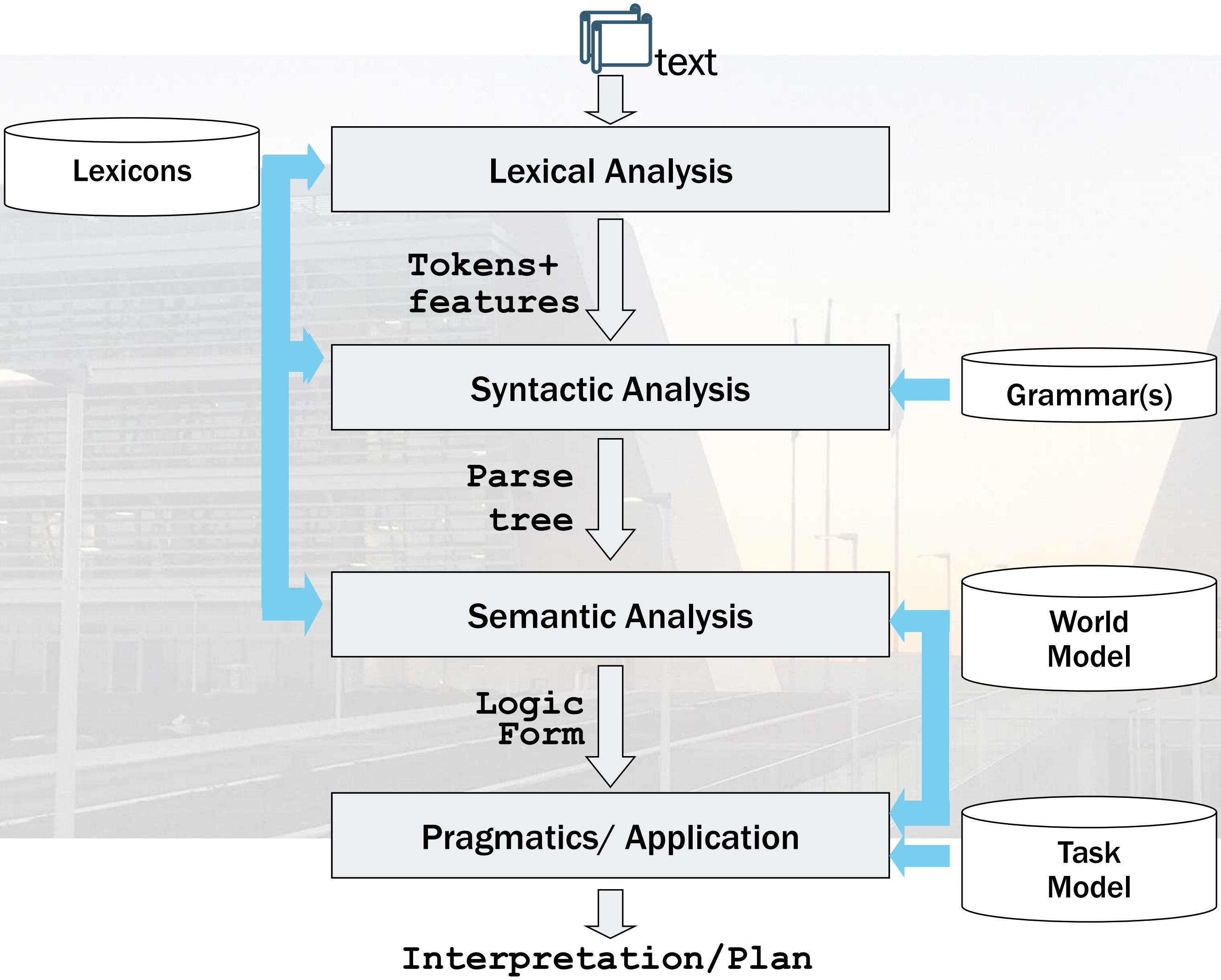
Kim was seen by Sam

John's son watched Kim

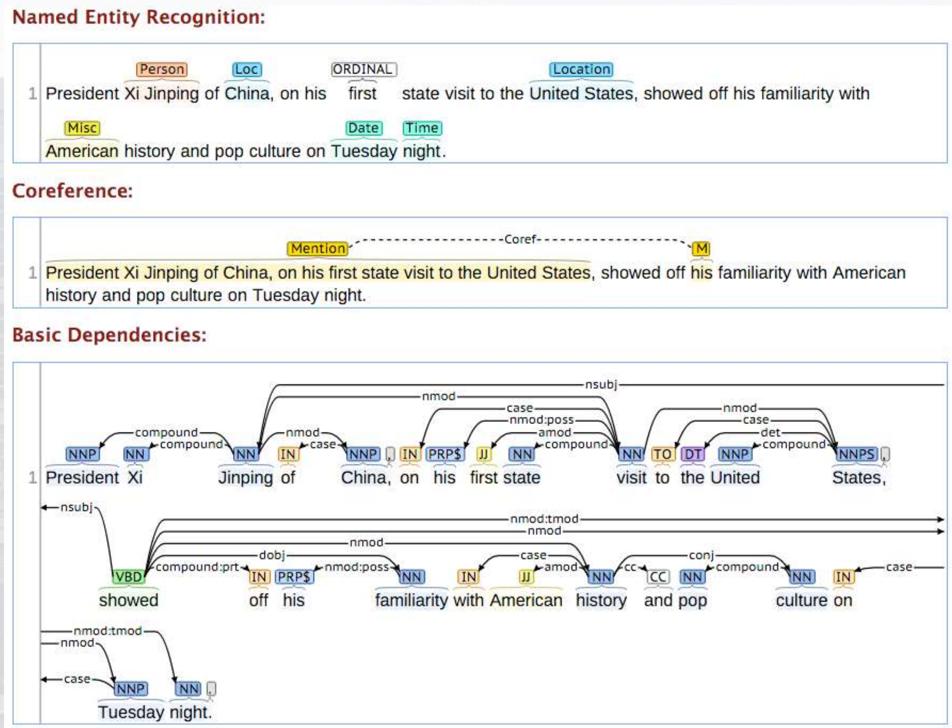
Sam saw Jane's daughter

```
saw(s,k)

?-saw(X,k).

X=s
```

NLP: THE STANDARD PROCESSING CHAIN (E.G. SPACY)



INTERPRETATION TASKS BEYOND PARSING: NAMED ENTITY RECOGNITION & COREFERENCE

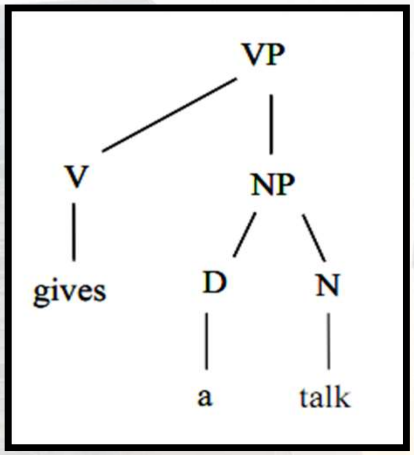


… GOING BACK TO LEARNING APPROACHES

KERNEL MACHINES FOR INTERPRETATION AND INFERENCE TASKS OVER LINGUISTIC DATA

Lesson 1: March 1°, 2023

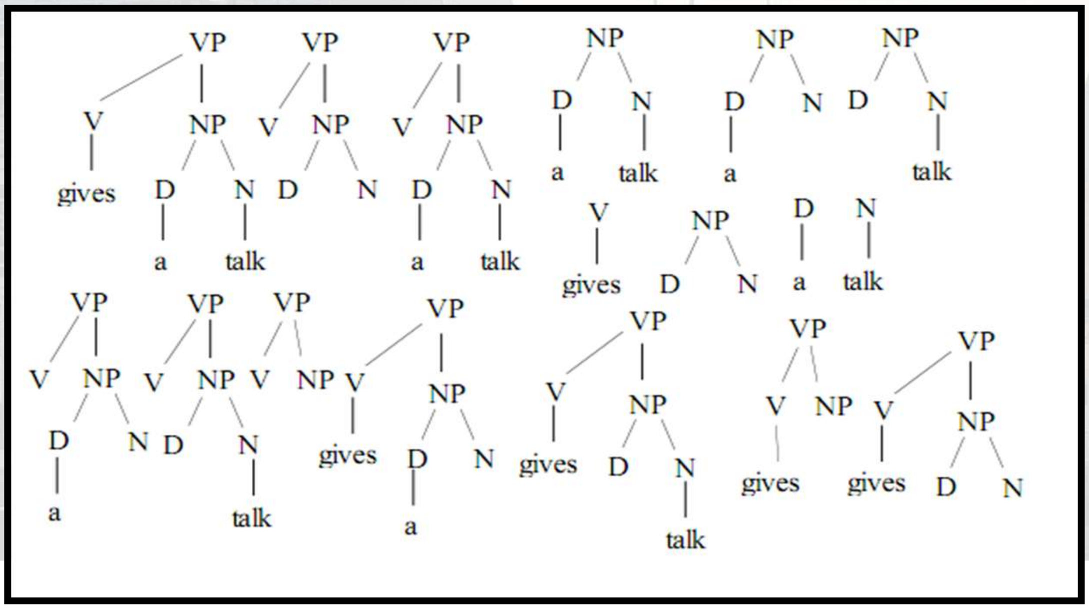
THE COLLINS AND DUFFY'S TREE KERNEL



Given a costituency tree

THE OVERALL FRAGMENT SET

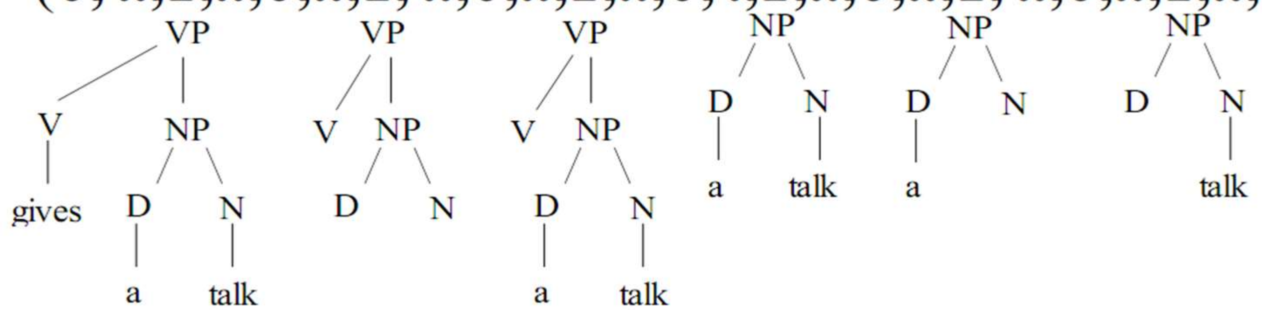
- We can explode the syntactic tree in all syntactically motivated fragments
- For each node the production rules must be respected, i.e. we can remove "0 or all children at a time"
- It is also known as Syntactic Tree Kernel



EXPLICIT FEATURE SPACE

- **Can we build a feature vector accounting on all this information?**

$$\vec{x} = (0, ..,1,..,0,..,1, ..,0,..,1,..,0, .,1,..,0,..,1, ..,0,..,1,..,0)$$



$\vec{x}_1 \cdot \vec{x}_2$ counts the number of common substructures

IMPLICIT REPRESENTATION

Can we estimate the tree kernel in an implicit space?

- We can implicitly count the number of common subtrees
- We prevent to define feature vectors that consider ALL POSSIBLE SUBTREES, i.e. thousand of features
- The final model will not contain feature vectors, but TREES

$$\vec{x}_1 \cdot \vec{x}_2 = \phi(T_1) \cdot \phi(T_2) = K(T_1, T_2) = \\ = \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} \Delta(n_1, n_2)$$

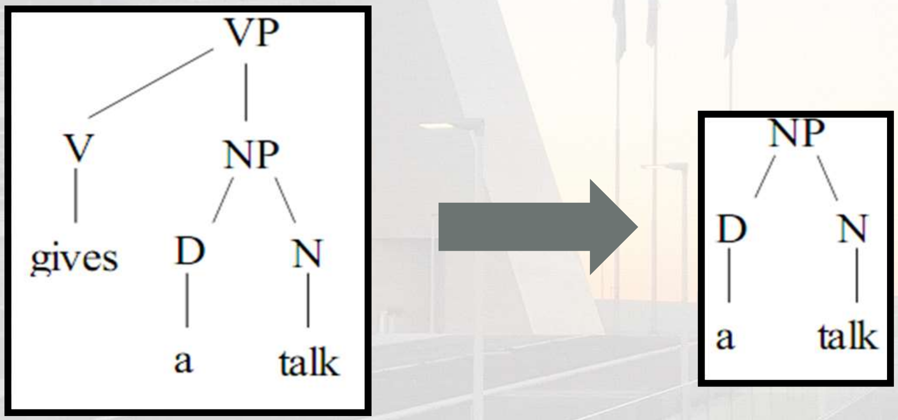
[Collins and Duffy, ACL 2002] evaluate Δ in $O(n^2)$:

$$\Delta(n_1, n_2) = 0, \ \textbf{if the productions are different else}$$
$$\Delta(n_1, n_2) = 1, \ \textbf{if pre-terminals else}$$
$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j)))$$

WEIGHTING IN GRAMMATICAL TREE KERNELS

In the kernel estimation different subtrees are taken in account different times

- Es: in the following trees, one fragment will contribute twice to the overall kernel



WEIGHTING

- A decay factor can be used, so the contribution of the embedded trees is reduced.
- The normalization of Tree Kernel estimation corresponds to the normalization of the explicit feature vector

Decay factor

$$\Delta(n_1, n_2) = \lambda, \quad \textbf{if pre-terminals else}$$

$$\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j)))$$

Normalization $$K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \times K(T_2, T_2)}}$$

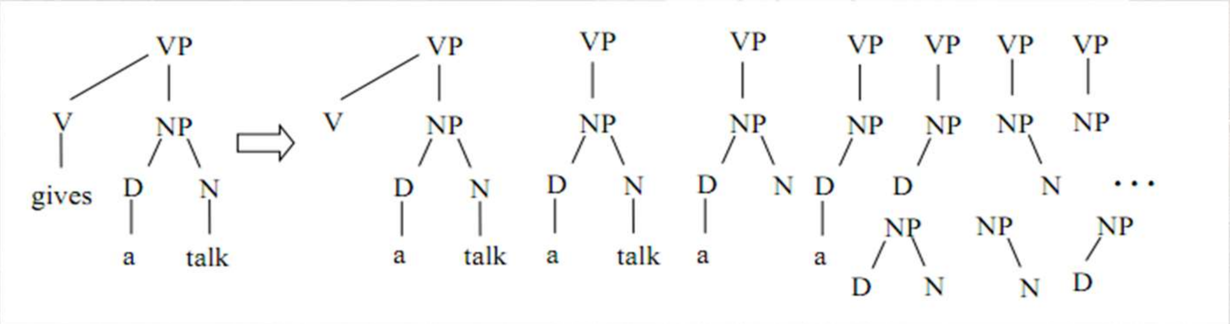
LEARNING UNDER KNOWLEDGE REPRESENTATION CONSTRAINTS

NATURE AND TYPES OF TREE KERNELS: C&D KERNEL, PARTIAL TREE KERNEL, COMPOSITIONALITY

PARTIAL TREE (MOSCHITTI, 2006)

- A Syntactic Tree satisfies completely a grammar rule, i.e. the constraint is "*remove 0 or all children at a time*".

- Partial Tree Kernel (PTK) relaxes such constraint we get more general substructures
  - It allows gaps in the production rules in the same fashion of the sequence kernel



PARTIAL TREE KERNEL

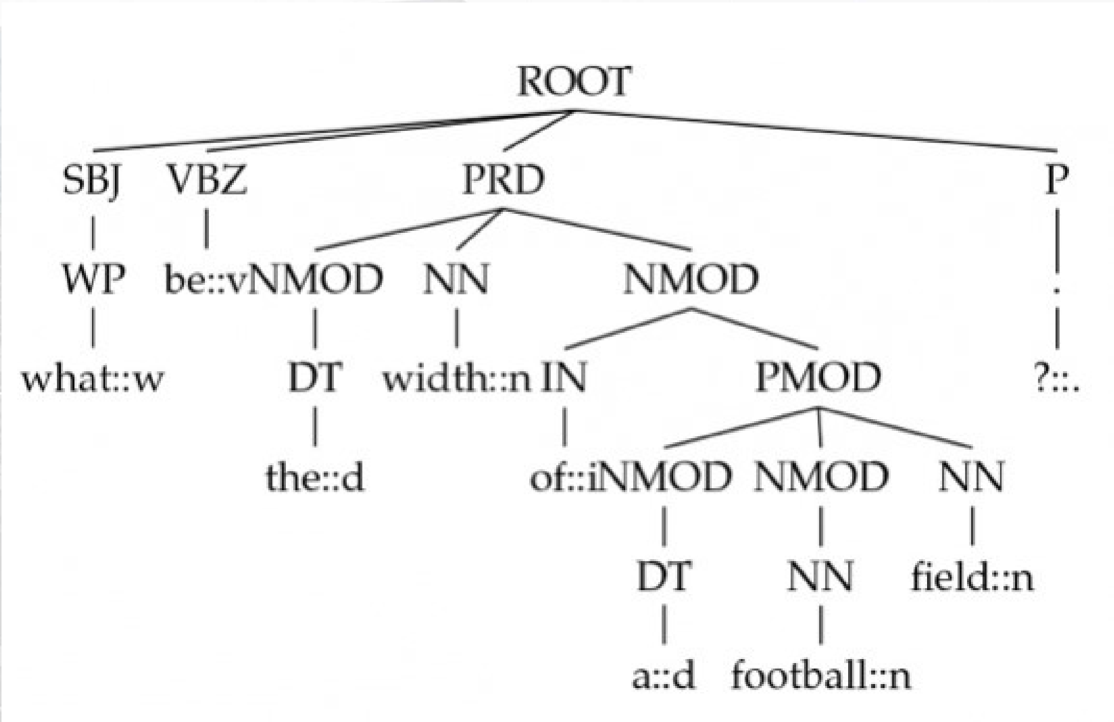
- if the node labels of n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
- else

$$\Delta(n_1, n_2) = 1 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1)=l(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}[\vec{J}_{1i}], c_{n_2}[\vec{J}_{2i}])$$

- By adding two decay factors we obtain:

$$\mu\left(\lambda^2 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1)=l(\vec{J}_2)} \lambda^{d(\vec{J}_1)+d(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}[\vec{J}_{1i}], c_{n_2}[\vec{J}_{2i}])\right)$$

GRAMMATICALLY CENTERED TREE KERNELS



LEXICALIZED TREE KERNELS

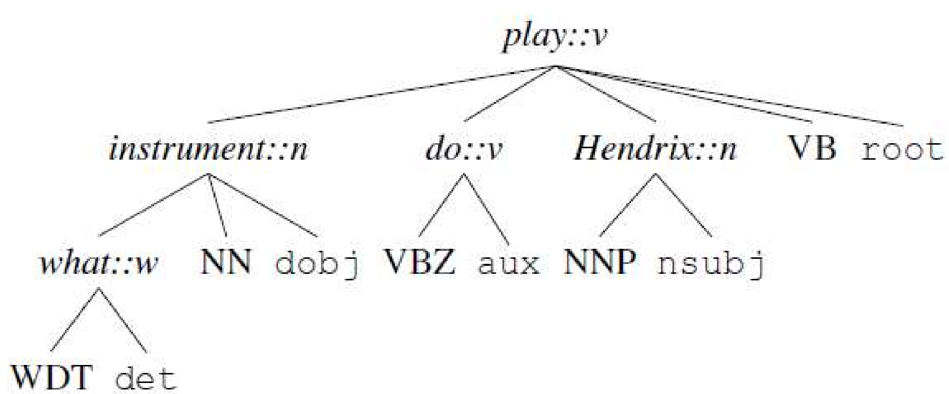


Figure 1: Lexical centered tree of the sentence "*What instrument does Hendrix play?*"

LEXICALIZED AND COMPOSITIONAL TREE KERNELS

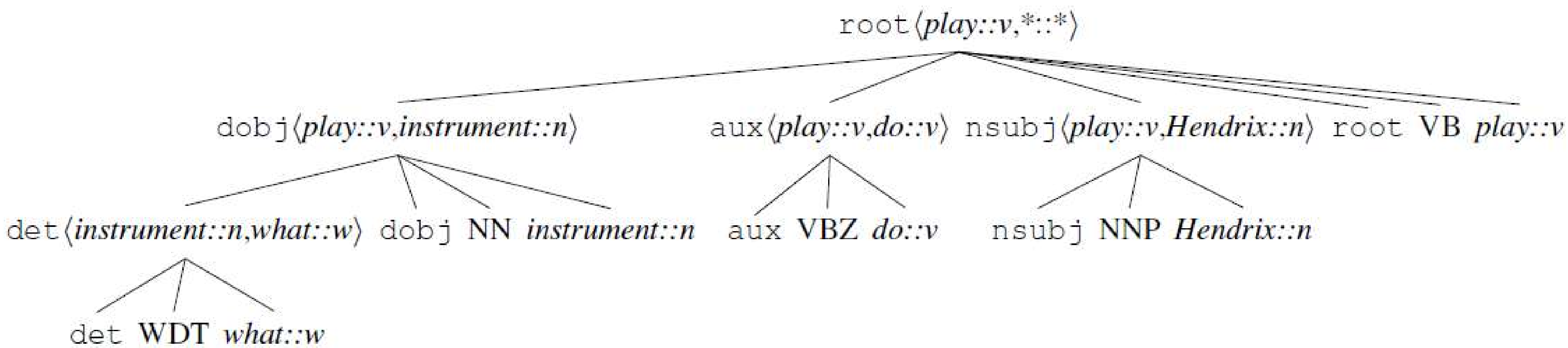


Figure 2: Compositional Lexical Centered Tree (CLCT) of the sentence "*What instrument does Hendrix play?*"

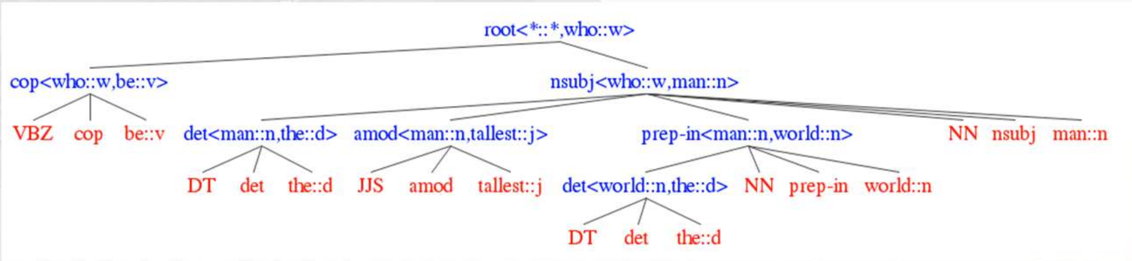
SEMANTIC SMOOTHING OF PTKs

$$\Delta_\sigma(n_1, n_2) = \mu\lambda\sigma(n_1, n_2), \text{ where } n_1 \text{ and } n_2 \text{ are leaves, else}$$

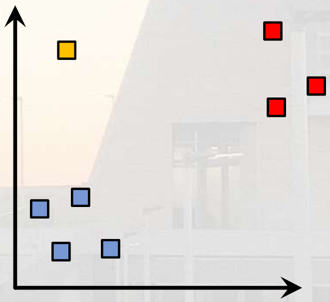
$$\Delta_\sigma(n_1, n_2) = \mu\sigma(n_1, n_2)\Big(\lambda^2 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1)=l(\vec{I}_2)} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta_\sigma(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))\Big)$$

TREE KERNELS ARE … EMBEDDING TOOLS

- Semantic Tree Kernels allows generating vectors that reflect syntactic/semantic information of sentences
  - *Who is the tallest man in the world?*



- Which most similar sentences/trees/vectors?
  - *Who is the richest woman in the world?*
  - *Who is the richest person in the world?*
  - *Who is the fastest swimmer in the world?*
  - *Who was murdered yesterday by the terrorist group?*
  - ….



COMPOSITIONALITY

$$\Delta_\sigma(n_1,n_2)=\mu\lambda\sigma(n_1,n_2), \text{where } n_1 \text{ and } n_2 \text{ are leaves, else}$$

$$\Delta_\sigma(n_1,n_2)=\mu\sigma(n_1,n_2)\Big(\lambda^2+\sum_{\vec{I}_1,\vec{I}_2,l(\vec{I}_1)=l(\vec{I}_2)} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)}\prod_{j=1}^{l(\vec{I}_1)}\Delta_\sigma(c_{n_1}(\vec{I}_{1j}),c_{n_2}(\vec{I}_{2j}))\Big)$$

- Tree nodes correspond to head-modifier pairs
- Individual contributions to the three kernels can be modeled as similarity scores in the (implict) embedding spaces
- First (m_1,h_1) and (m_2,h_2) pairs are mapped into the space, and then the similairty at each node is computed as a combination of the cosine similarity estimates in the suitable subspaces (Annesi et al, CIKM 2014)

COMPOSITIONALLY SMOOTHED PARTIAL TREE KERNEL

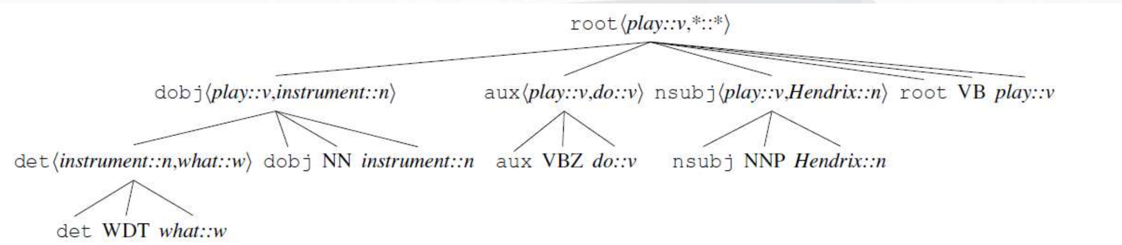


Figure 2: Compositional Lexical Centered Tree (CLCT) of the sentence "*What instrument does Hendrix play?*"

Algorithm 1 $\sigma_\tau(n_x, n_y, lw)$ Compositional estimation of the lexical contribution to semantic tree kernel

$\sigma_\tau \leftarrow 0,$
*/*Matching between simple lexical nodes*/*
if $n_x = \langle lex_x{::}pos \rangle$ **and** $n_y = \langle lex_y{::}pos \rangle$ **then**
  $\sigma_\tau \leftarrow \sigma_{LEX}(n_x, n_y)$
end if
*/*Matching between identical grammatical nodes, e.g. POS tags*/*
if $(n_x = pos$ **or** $n_x = dep)$ **and** $n_x = n_y$ **then**
  $\sigma_\tau \leftarrow lw$
end if
if $n_x = \langle d_{h,m}, \langle li_x \rangle \rangle$ **and** $n_y = \langle d_{h,m}, \langle li_y \rangle \rangle$ **then**
  */*Matching between compositional nodes: both modifiers are missing*/*
  if $li_x = \langle h_x{::}pos \rangle$ **and** $li_y = \langle h_y{::}pos \rangle$ **then**
    $\sigma_\tau \leftarrow \sigma_{Comp}((h_x), (h_y)) = \sigma_{LEX}(n_x, n_y)$
  end if
  */*Matching between compositional nodes: one modifier is missing*/*
  if $li_x = \langle h_x{::}pos_h \rangle$ **and** $li_y = \langle h_y{::}pos_h, m_y{::}pos_m \rangle$ **then**
    $\sigma_\tau \leftarrow \sigma_{Comp}((h_x, h_x), (h_y, m_y))$
  end if
  */*Matching between compositional nodes: the general case*/*
  if $li_x = \langle h_x{::}pos_h, m_x{::}pos_m \rangle$ **and** $li_y = \langle h_y{::}pos_h, m_y{::}pos_m \rangle$ **then**
    $\sigma_\tau \leftarrow \sigma_{Comp}((h_x, m_x), (h_y, m_y))$
  end if
end if
return σ_τ