依存句法分析介绍

常宝宝
北京大学计算语言学研究所
chbb@pku.edu.cn

语法理论概要

- 存在不同的语法理论描写句子的句法结构。
- 短语结构语法(Phrase structure grammar)
  描写句子成分之间的组成关系(constituency relation)
  也叫成分语法(constituency grammars)
- 依存语法(Dependency grammar)
  描写句中词和词之间的依存关系(dependency relation)
- 基于CFG的句法分析是面向短语结构语法理论的句法分析。

依存语法理论

- 法国语言学家Lucien Tesnière(1893-1954)最早提出用句中词和词之间的依存关系描述句子的句法结构。(《结构句法基础》，1959年)
- 与短语结构语法一样，也有许多现代语法理论建立在依存语法的基础上。如 Functional Generative Description(1986)、Meaning-Text Theory(1988)、word grammar(1990)等。
- NLP中，有大量面向依存语法的句法分析研究，即依存句法分析(Dependency Parsing)研究。

依存关系(dependency relation)

- 句子中词和词之间存在一种从属和中心的关系。如：
  木头·**桌子**　financial·**Markets**
  吃·包子　　**buy**·books
  …
- 处于中心地位的词，称为**中心词**(head)，处于从属地位的词称为**从属词**(dependent)。
  - head 有时也称作governor、regent
  - dependent 有时也称作 modifier
- 依存关系是一种二元非对称关系，可以被进一步分成多种类型，如：
  木头 和 桌子 之间是一种修饰关系，标记为 NMOD
  吃 和 包子 之间是一种动词宾语关系，标记为DOBJ

句子的依存结构(dependency structure)

- 把词视作结点，依存关系视作边(或称弧)，句子结构可表示成带标记的有向图，即依存图(dependency graph)。

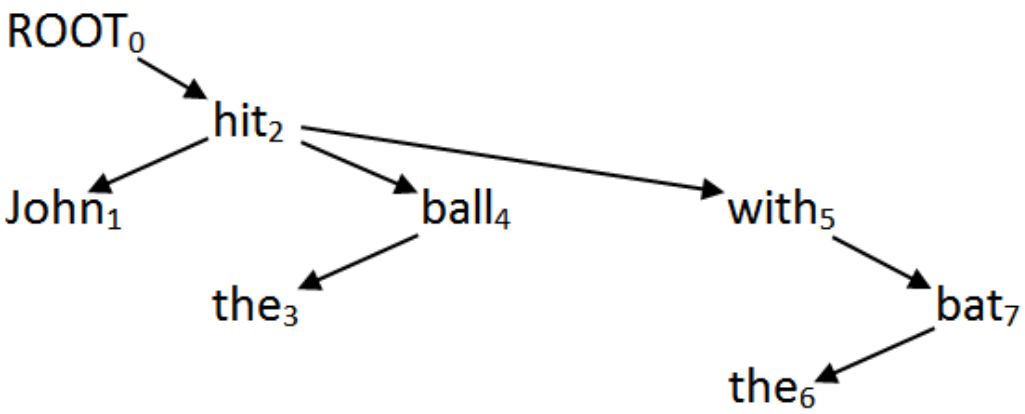
- 通常依存图遵循如下限制：
  - **连通性限制**：句子的依存表示构成连通图。(connectivity)
  - **单中心限制**：句子中的词最多只有一个中心与之连接。(single-head)
  - **非环限制**：句子依存图中不能有环。(acyclicity)

  可以证明，符合上述限制的依存图是一棵有根树，即**依存树**(dependency tree)。只有根结点没有与之关联的中心词。为了处理的方便，通常人为添加一个特殊的词ROOT作为根结点的中心词。

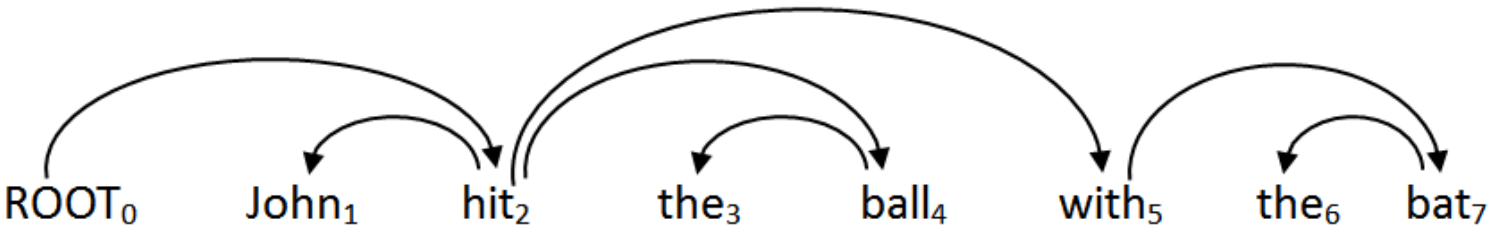
依存树表示

John hit the ball with the bat

①



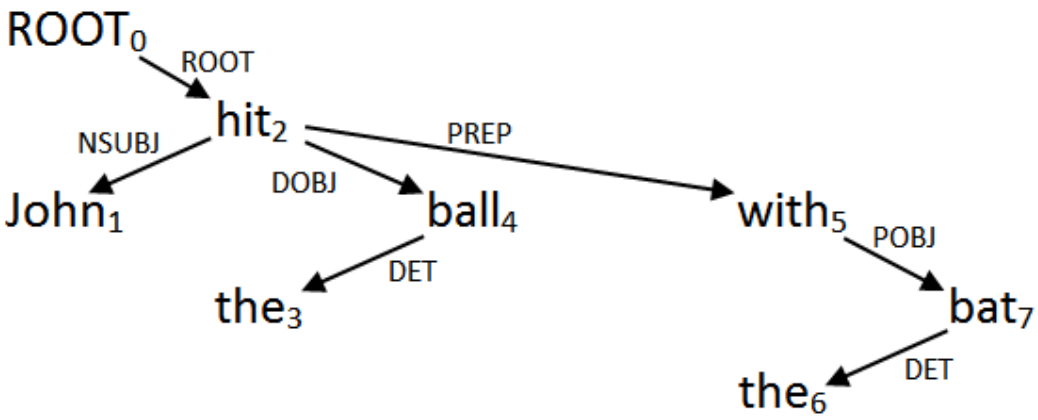
②



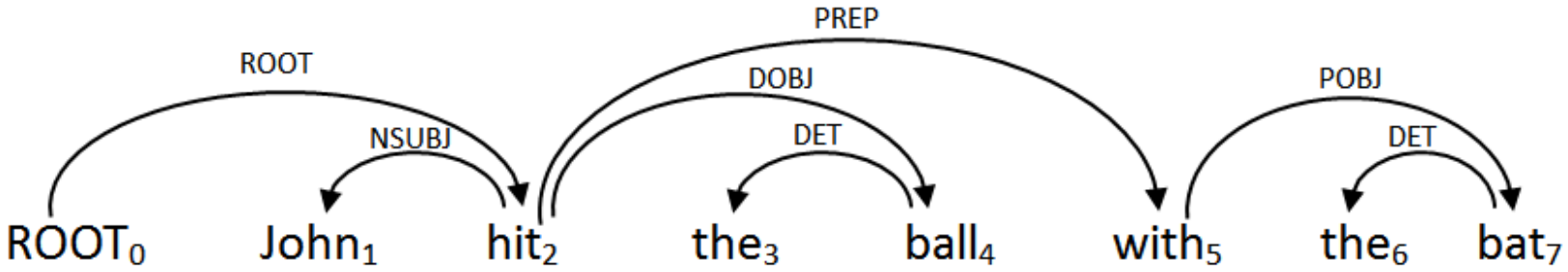
依存树表示

John hit the ball with the bat

①



②



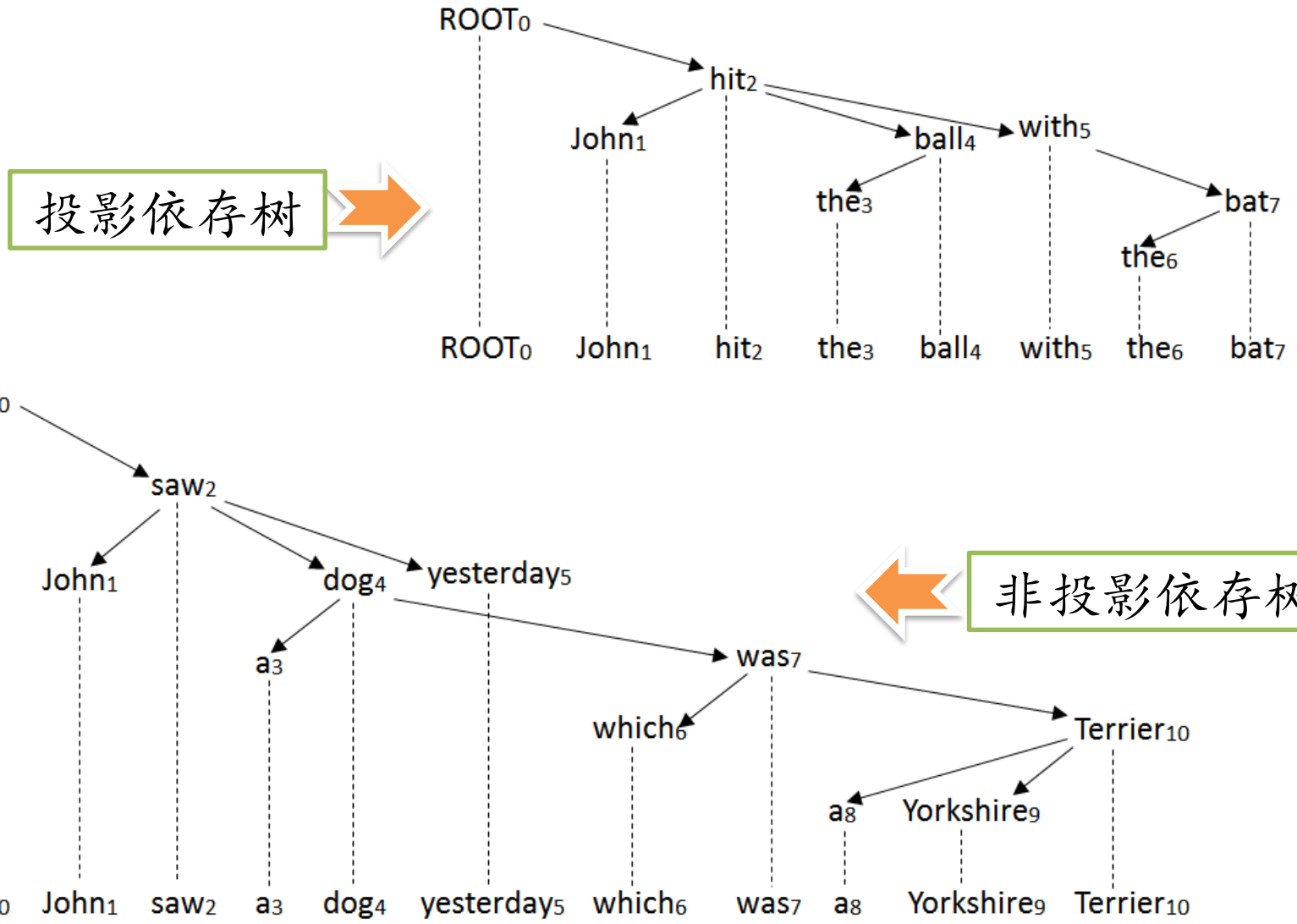
投影性(projectivity)

- 依存弧可以表示为三元组(head, label, dependent)，例如：

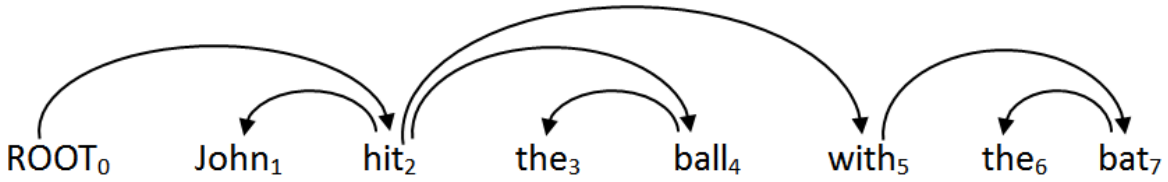
  (hit, NSUBJ, John)　　(hit, DOBJ, ball)　(hit, PREP, with)
  (ball, DET, the)(with, POBJ, bat)　(bat, DET, the)

- 令w_i为依存树中的结点，称w_i沿着依存弧可到达的结点(词)为w_i的**投影**(projection)。
- 投影性条件：若w_i在依存树中的所有投影 对应句子中的连续片段，则称该结点满足投影性条件。
- 若依存树中所有结点满足投影性条件，则依存树称为**投影依存树**(projective dependency tree)，否则称为**非投影依存树**(non-projective dependency tree)。

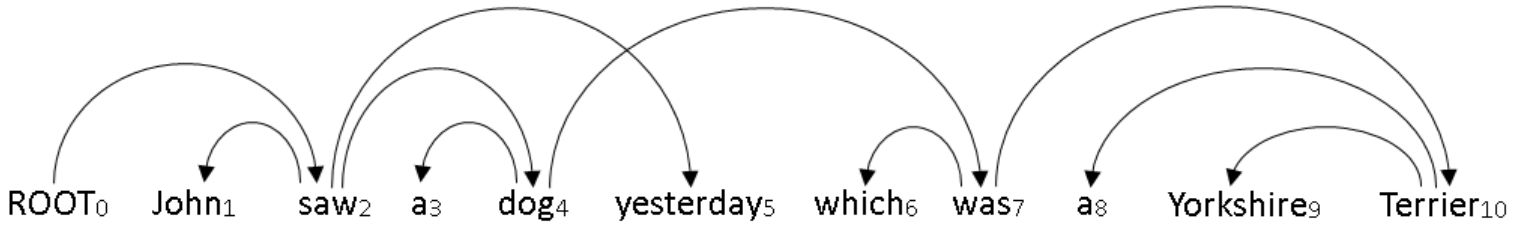
投影性


投影依存树
ROOT0 hit2 John1 ball4 with5 the3 bat7 the6
ROOT0 John1 hit2 the3 ball4 with5 the6 bat7
非投影依存树
ROOT0 saw2 John1 dog4 yesterday5 a3 was7 which6 Terrier10 a8 Yorkshire9
ROOT0 John1 saw2 a3 dog4 yesterday5 which6 was7 a8 Yorkshire9 Terrier10

投影性



投影依存树



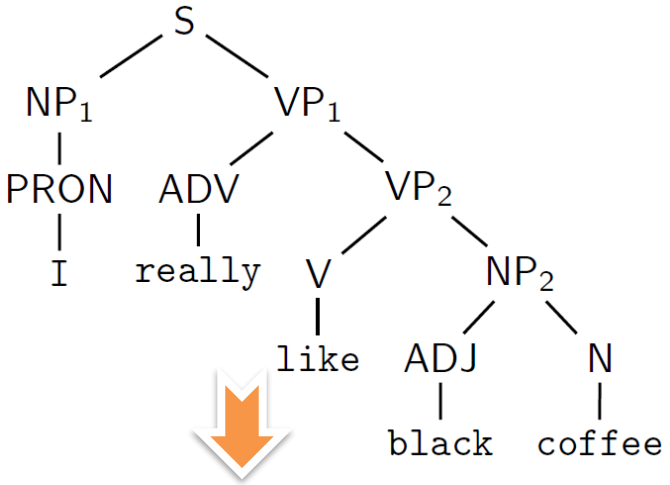
非投影依存树

依存语法...

- 依存语法是一种词语法。
- 依存结构中没有短语(成分)的概念。
- 依存结构表示比短语结构表示简练。
- 依存结构与短语结构存在大致的转换关系。
  - 许多依存树库通过短语结构树库转换生成
    如：通过Penn Treebank 生成英语或者汉语的依存树库。
- 通常认为依存语法更适合描述语序自由的语言。
  - 德语、捷克语、荷兰语、土耳其语等
- 语言中大部分结构是投影结构。
- 语序自由的语言中含有较多的非投影结构。
  - 汉语、英语中的非投影结构较少。

从短语结构树到依存树

- 短语结构树中通常不会标记词和词之间的依存关系。
- 可以通过启发式规则(heuristics)确定成分的中心(head)成分。
- 中心成分规则(Head rules)
  给定：$X \rightarrow Y_1 Y_2 \cdots Y_n$
  怎样确定中心成分Y_i?



短语类型	搜索方向	中心成分优先列表
S	right-to-left	*Aux VP NP AP PP*
VP	left-to-right	*V VP*
NP	right-to-left	*Pron N NP*

短语	中心成分	中心词
S	VP_1	like
VP_1	VP_2	like
VP_2	*V*	like
NP_1	*Pron*	I
NP_2	*N*	coffee

从短语结构树到依存树

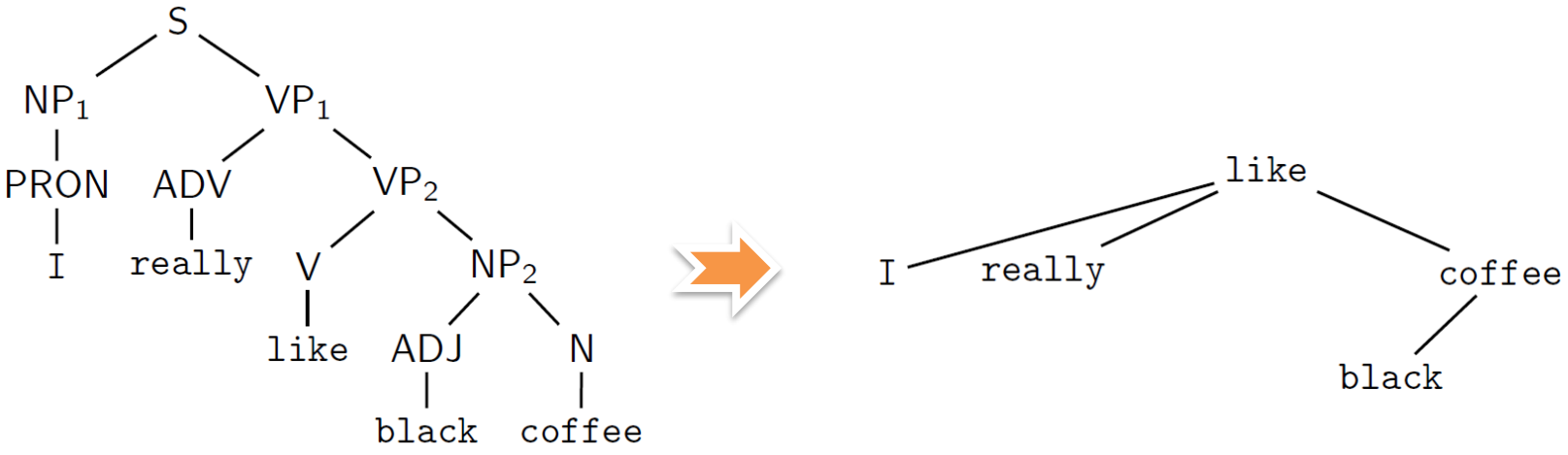
- 让成分的其他组成成分依存中心成分。

  对于S，NP_1 依存 VP_1，I ← like

  对于VP_1，ADV 依存 VP_2，really ← like

  对于VP_2，NP_2 依存 V，like → coffee

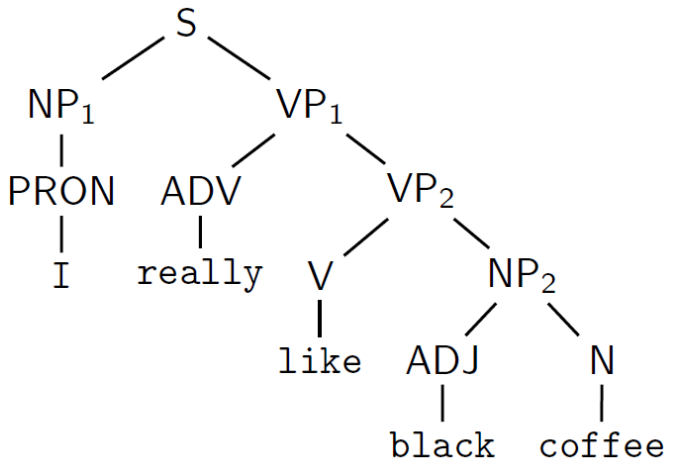
  对于NP_2，ADJ 依存 N，black ← coffee

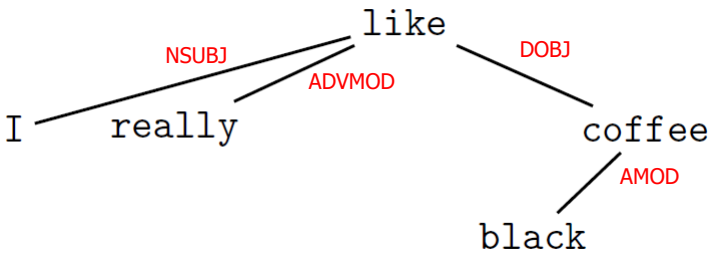


从短语结构树到依存树

- 利用成分支配关系确定功能标记。
  给定$X \rightarrow Y_1 Y_2 \cdots Y_n$ 且 $headword(Y_h) \rightarrow headword(Y_d)$
  可依据成分支配关系(Y_h, X, Y_d)确定相应依存关系。

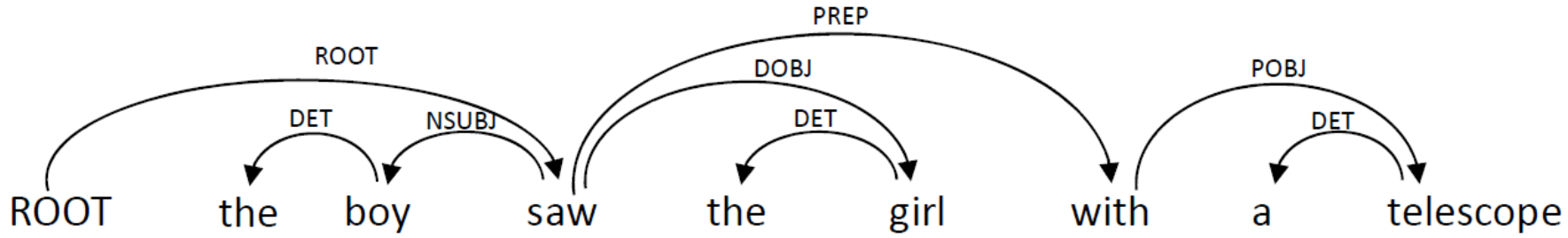
依存关系 ⇨	成分支配关系 ⇨	依存关系标记
I←like	(NP_1, S, VP_1)	NSUBJ
really←like	(ADV, VP_1,VP_2)	ADVMOD
like→coffee	(V, VP_2, NP_2)	DOBJ
black←coffee	(ADJ, NP_2, N)	AMOD

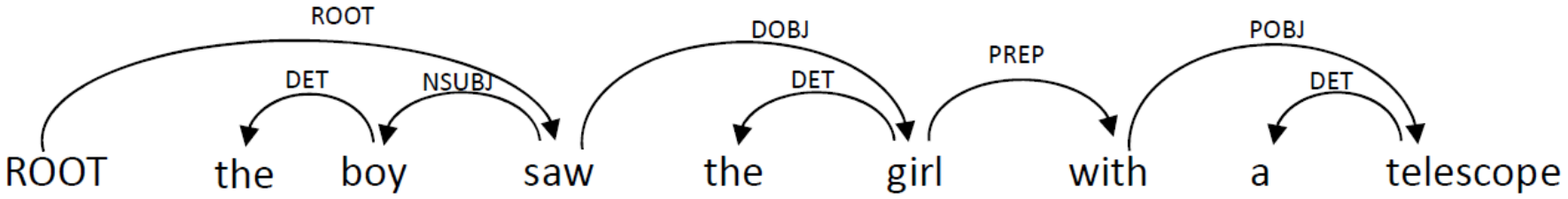




句法歧义

the boy saw the girl with a telescope





依存句法分析

- 对给定的自然语言句子，分析并得到其依存句法树。
- 依存句法分析的主要方法
  - 基于图的依存分析(Graph-based dependency parsing)
  - 基于转移的依存分析(Transition-based dependency parsing)
- 模型：寻找得分最高的依存树

$$\hat{y} = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}}\, s(x, y)$$

x是待分析的句子，$\mathcal{Y}(x)$是所有可能的依存树，$s(x, y)$是依存树y的得分。

基于图的依存分析

- 基于图的依存分析
  - 评分模型(scoring model)
    - 如何计算依存树的得分$s(x, y)$？
  - 分析算法(parsing algorithm)
    - 给定评分模型，如何搜索得分最高的依存树？
- 主要思想
  - 将依存树分解为子图(用c 表示)，子图独立评分，依存树得分定义为子图得分之和

$$s(x, y) = \sum_{c \in y} s(x, c)$$

弧-分解模型

- 如何定义子图？
  存在不同做法，最简单的是弧-分解模型(arc-factored model)。
- 弧-分解模型（一阶模型）
  - 子图定义为依存边(弧)，依存树分解为依存边的集合
  - 依存树的得分定义为所有依存边的得分之和

$$s(x,y) = \sum_{(i,j)\in y} s(i,j)$$

(i,j)为树中的依存边，即依存树中存在依存边$x_i \rightarrow x_j$。

  - 如何计算依存边的得分？
    依存边的得分通常定义为一个特征向量与权重向量的内积，即

$$s(i,j) = \mathbf{w} \cdot \mathbf{f}(i,j)$$

    注意: $s(i,j) \neq s(j,i)$

特征选择和参数学习

- 计算边(i,j)的得分，需要针对依存边提取特征
  $pos(i) = verb$ //中心词是动词
  $pos(j) = noun$ //依存词是名词
  $pos(i_{-1}) = noun$ //中心词前一个词是名词
  $pos(i) = verb \;\&\&\; pos(j) = noun$
  $pos(i) = verb \;\&\&\; pos(i_{-1}) = noun$
  $direction(i,j) = \leftarrow$
  $distance(i,j) = number$
  …
- 权重向量$\mathbf{w}$是模型参数，需通过依存树库训练得到。
  - 通常通过结构化学习技术学习参数。
  - Structured perceptron, Margin Infused Relaxed Algorithm (MIRA); online learning

参数学习

- 结构化参数学习基本思想
  - 迭代训练
  - 给定句子x及其标准依存树y(gold-standard)
  - 根据当前参数$\mathbf{w}$,计算得分最高的依存树y'
  - 增加出现在y中但未出现在y'中的特征对应的权重
  - 降低出现在y'中但未出现在y中的特征对应的权重

Eisner算法

- Eisner算法是一种自底而上的分析算法。
- Eisner是基于表的分析算法。是动态规划算法。
- 在Eisner算法中，表的单元格$C[s,t]$中存放的是句子片段$w_s w_{s+1} \cdots w_t$对应的最优依存树(片段)。
- Eisner算法中的span不是句子中的任意子串，整个span的中心词必须位于span的边界处，即只能是w_s或者w_t。

Eisner算法

- span内部的词不存在与span外部词的依存连接。
- span分为两种类型：complete span和incomplete span

  1) complete span:

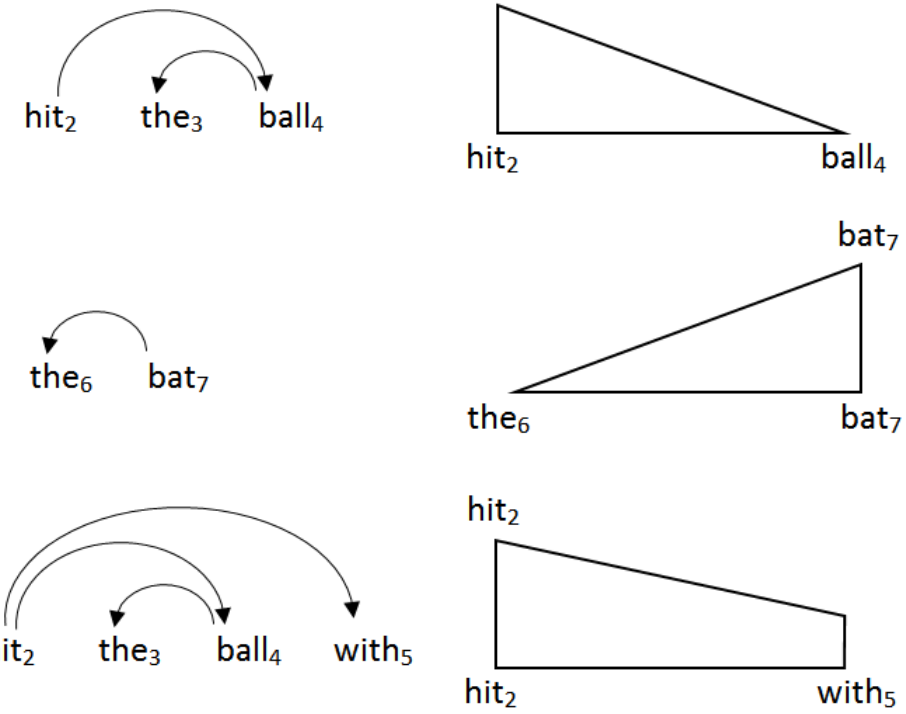
  除了span的中心词，span中的其他词都已经完成分析。也就是其他词的中心词和所有从属词均已确定。

  2) incomplete span

  除了span两端的词(即w_s和w_t)，span中的其他词都已经完成分析。

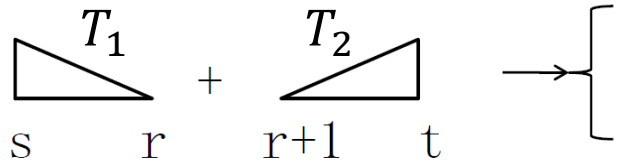
Eisner算法

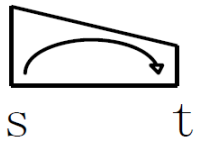
- span举例和图形表示



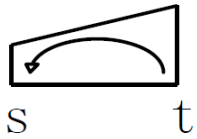
Eisner算法

- Eisner算法基本操作
  - 生成incomplete span



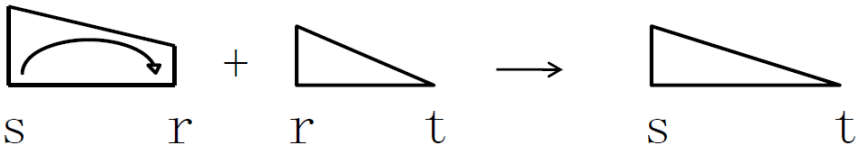


$T \quad s(T) = s(T_1) + s(T_2) + s(s,t)$

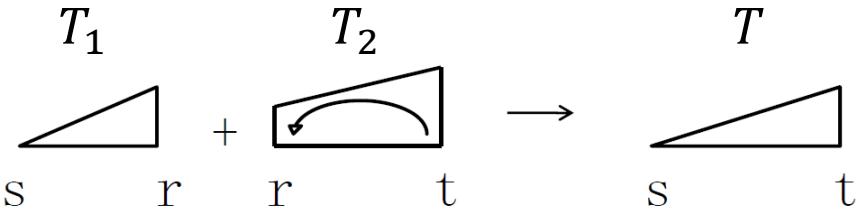


$T \quad s(T) = s(T_1) + s(T_2) + s(t,s)$

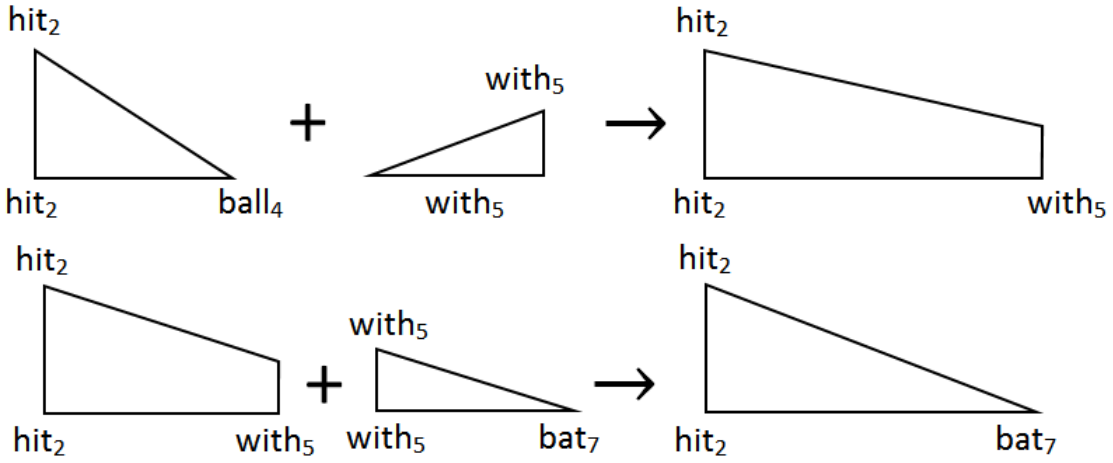
  - 生成complete span

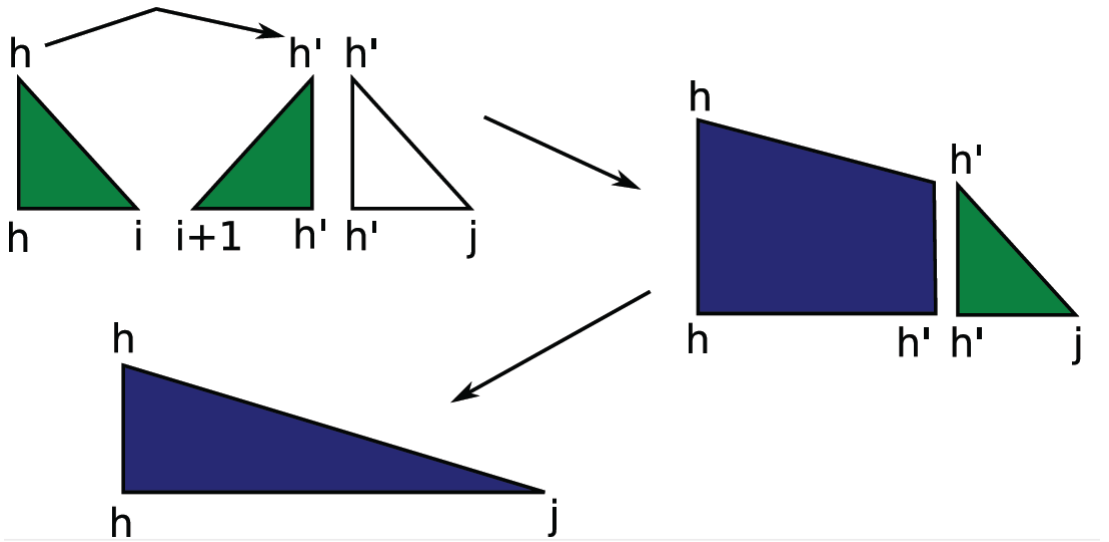


$s(T) = s(T_1) + s(T_2)$



Eisner算法





因为head位于span的边界，Eisner算法的复杂度为$O(n^3)$。

Eisner算法

- Span的表示是一个四元组(s, t, d, c)
  - s和t表示span的起止位置，
  - d表示head位置，left或right
  - c表示完成状态，complete或incomplete

  $(hit_2, with_5, \rightarrow, incomplete)$

  $(hit_2, bat_7, \rightarrow, complete)$
- 数据结构$C[s][t][d][c]$
- 自底而上填表，依次增加处理的span的长度。
- 算法初始化：长度为1的span，初始化两个complete span，分别为左中心和右中心，score为0。

Eisner算法

Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$

for $k: 1..n$

  for $s: 1..n$

    $t = s + k$

    if $t > n$ then break

    $C[s][t][\leftarrow][0] = \max_{s \le r < t}(C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s)$

    $C[s][t][\rightarrow][0] = \max_{s \le r < t}(C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t)$

    $C[s][t][\leftarrow][1] = \max_{s \le r < t}(C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0]$

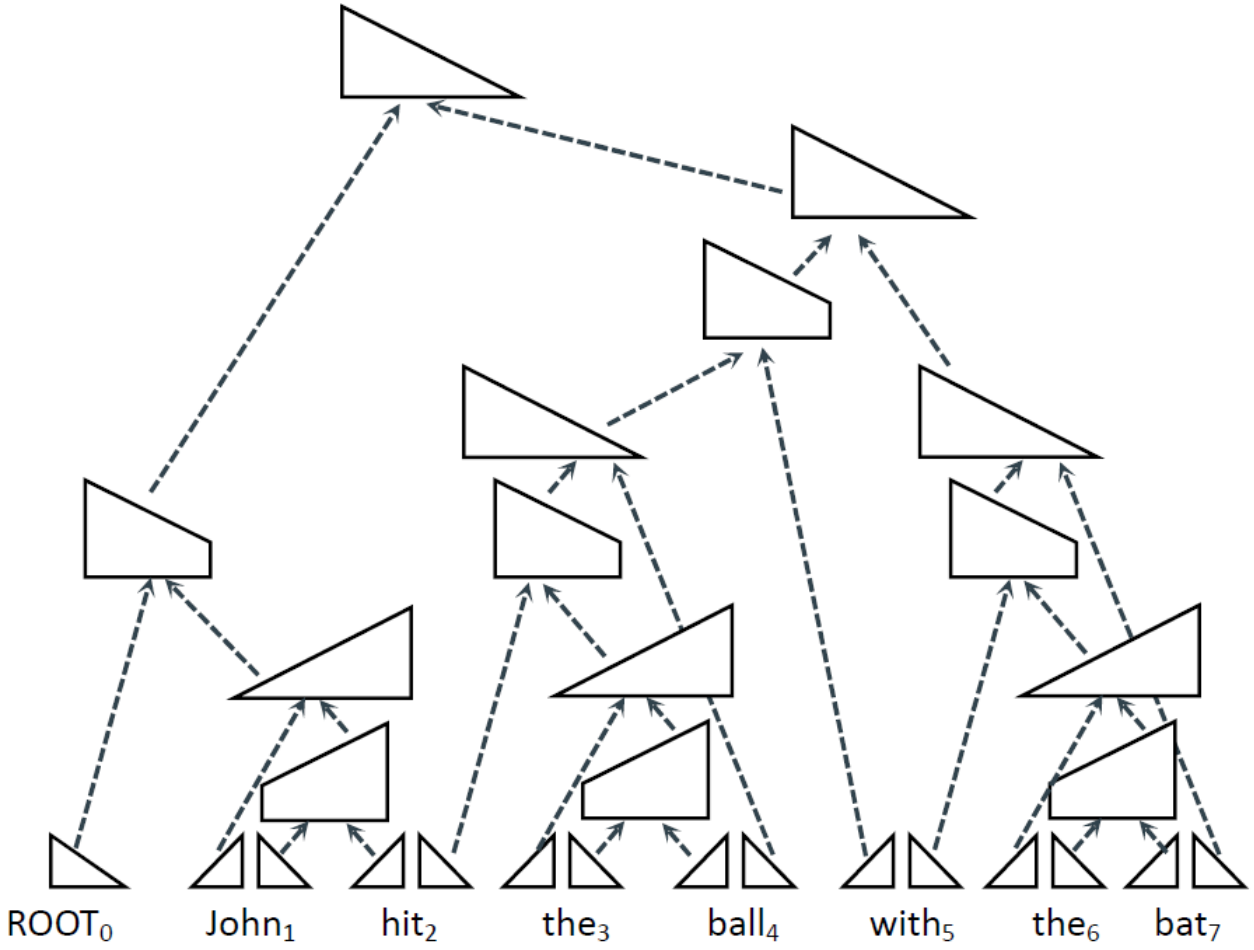
    $C[s][t][\rightarrow][1] = \max_{s < r \le t}(C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1]$

  end for

end for

return $C[0][n][\rightarrow][1]$

Eisner算法



Eisner算法

- Eisner算法的时间复杂度:$O(n^3)$
- Eisner算法只能生成投影依存树
- 增加依存关系标记(s, t, l, d, c)和记录树结构
- 对于基于图的依存分析而言，Eisner算法是基础算法。
- 高阶模型，代价是时间复杂度提高

分析算法

- 最大生成树(Maximum Spanning Tree)算法
- 给定句子x,构造带权有向图$G_x = (V_x, E_x)$,其中：
  - 结点集合： $V_x = \{x_0 = \mathrm{ROOT}, x_1, x_2, \cdots, x_n\}$
  - 有向边集合： $E_x = \{(i,j): x_i \neq x_j, x_i \in V_x, x_j \in V_x - \mathrm{ROOT}\}$
  - 边(i,j)的权值为$s(i,j)$
- G_x的**生成树**$G'_x = (V'_x, E'_x)$, 其中：
  - $V'_x = V_x$, $E'_x \subset E_x$
  - 结点ROOT的入度为0，其余结点的入度为1
- G_x的**最大生成树**：G_x生成树中权值最大的树。
- 投影依存树，最大生成树需满足投影性条件
  - Eisner算法。
- 非投影依存树：朱-刘算法(Chu-Liu-Edmond Algorithm)

基于转移的依存分析

- 在短语结构句法分析中，常采用移入-归约(shift-reduce)策略进行句法分析。依存句法分析，也可以采用类似的策略。
- 依存树是通过执行一个分析动作序列逐步构造出来的。
- 句法分析始于一个初始格局(configuration)，应用分析动作后，产生新的格局，再应用分析动作，又产生更新的格局，……。
- 分析过程表现为一种格局转移过程，所以称为基于转移的依存分析。（transition-based dependency parsing）

基于转移的依存分析

- 待分析的句子为 $x = x_0 x_1 \cdots x_n$
- **分析格局**$c = (\sigma, \beta, A)$，其中：
  - σ是分析栈，存放处理中的结点(词)。
  - β是输入缓冲区，存放尚未处理的结点(词)。
  - A是依存弧的集合，代表分析过程中得到的部分依存树。
- **基于转移的系统**是一个四元组$S = (C, T, c_s, C_t)$，其中：
  - C是分析格局的集合
  - T是分析动作的集合
  - c_s是初始格局，通常$c_s(x) = ([x_0], [x_1 x_2 \cdots x_n], \emptyset)$
  - C_t是终止格局集合，通常$C_t = \{c | c = (\sigma, [], A), c \in C\}$

基于转移的依存分析

- 栈与缓冲区的表示
  - 栈：$\sigma|i$　　i是栈顶元素、σ为其余元素
  - 缓冲区：$j|\beta$　　j是缓冲区头部元素、β为其余元素
- 基于转移的依存分析算法
  - arc-standard分析算法
  - arc-eager分析算法
- 两种分析算法支持不同的分析动作
  - arc-standard算法：三种分析动作
  - arc-eager算法：四种分析动作

arc-standard分析算法

Transitions

LEFT-ARC$_l$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma,j\|\beta,A\cup\{(j,l,i)\})$
RIGHT-ARC$_l^s$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma,i\|\beta,A\cup\{(i,l,j)\})$
SHIFT	$(\sigma,i\|\beta,A) \Rightarrow (\sigma\|i,\beta,A)$

Preconditions

LEFT-ARC$_l$	$\neg[i=0]$ $\neg\exists k\exists l'[(k,l',i)\in A]$
RIGHT-ARC$_l^s$	$\neg\exists k\exists l'[(k,l',j)\in A]$

l 是依存关系标记。

LEFT-ARC 和 RIGHT-ARC分别添加左向和右向的依存弧。

SHIFT将缓冲区中的词移入分析栈。

arc-standard分析实例

分析动作		分析格局		
		分析栈	缓冲区	依存树
		$([\text{ROOT}_0]$,	$[\text{John}_1 \ldots \text{bat}_7]$,	$A_0 = \emptyset)$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0\ \text{John}_1]$,	$[\text{hit}_2 \ldots \text{bat}_7]$,	$A_0)$
$\text{LEFT-ARC}_{\text{NSUBJ}}$	$\Longrightarrow$	$([\text{ROOT}_0]$,	$[\text{hit}_2 \ldots \text{bat}_7]$,	$A_1 = \{(\text{hit}_2, \text{NSUBJ}, \text{John}_1)\})$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2]$,	$[\text{the}_3 \ldots \text{bat}_7]$,	$A_1)$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2\ \text{the}_3]$,	$[\text{ball}_4 \ldots \text{bat}_7]$,	$A_1)$
$\text{LEFT-ARC}_{\text{DET}}$	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2]$,	$[\text{ball}_4 \ldots \text{bat}_7]$,	$A_2 = A_1 \cup \{(\text{ball}_4, \text{DET}, \text{the}_3)\})$
$\text{RIGHT-ARC}_{\text{DOBJ}}$	$\Longrightarrow$	$([\text{ROOT}_0]$,	$[\text{hit}_2 \ldots \text{bat}_7]$,	$A_3 = A_2 \cup \{(\text{hit}_2, \text{DOBJ}, \text{ball}_4\})$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2]$,	$[\text{with}_5 \ldots \text{bat}_7]$,	$A_3)$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2\ \text{with}_5]$,	$[\text{the}_6\ \text{bat}_7]$,	$A_3)$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2\ \text{with}_5\ \text{the}_6]$,	$[\text{bat}_7]$,	$A_3)$
$\text{LEFT-ARC}_{\text{DET}}$	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2\ \text{with}_5]$,	$[\text{bat}_7]$,	$A_4 = A_3 \cup \{(\text{bat}_7, \text{DET}, \text{the}_6)\})$
$\text{RIGHT-ARC}_{\text{POBJ}}$	$\Longrightarrow$	$([\text{ROOT}_0\ \text{hit}_2]$,	$[\text{with}_5]$,	$A_5 = A_4 \cup \{(\text{with}_5, \text{POBJ}, \text{bat}_7\})$
$\text{RIGHT-ARC}_{\text{PREP}}$	$\Longrightarrow$	$([\text{ROOT}_0]$,	$[\text{hit}_2]$,	$A_6 = A_5 \cup \{(\text{hit}_2, \text{PREP}, \text{with}_5\})$
$\text{RIGHT-ARC}_{\text{ROOT}}$	$\Longrightarrow$	([],	$[\text{ROOT}_0]$	$A_7 = A_6 \cup \{(\text{ROOT}_0, \text{ROOT}, \text{hit}_2)\})$
SHIFT	$\Longrightarrow$	$([\text{ROOT}_0]$,	[]	A_7

arc-eager分析算法

Transitions

LEFT-ARC$_l$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma,j\|\beta,A\cup\{(j,l,i)\})$
RIGHT-ARC$_l^e$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma\|i\|j,\beta,A\cup\{(i,l,j)\})$
REDUCE	$(\sigma\|i,\beta,A) \Rightarrow (\sigma,\beta,A)$
SHIFT	$(\sigma,i\|\beta,A) \Rightarrow (\sigma\|i,\beta,A)$

Preconditions

LEFT-ARC$_l$	$\neg[i=0]$ $\neg\exists k\exists l'[(k,l',i)\in A]$
RIGHT-ARC$_l^e$	$\neg\exists k\exists l'[(k,l',j)\in A]$
REDUCE	$\exists k\exists l[(k,l,i)\in A]$

- LEFT-ARC、SHIFT与arc-standard算法相同
- RIGHT-ARC不同
- 增加REDUCE动作

arc-eager算法分析实例

分析动作		分析格局		
		分析栈	缓冲区	依存树
		([ROOT$_0$],	[John$_1$... bat$_7$],	$A_0 = \emptyset$)
SHIFT	$\Longrightarrow$	([ROOT$_0$ John$_1$],	[hit$_2$... bat$_7$],	A_0)
LEFT-ARC$_{NSUBJ}$	$\Longrightarrow$	([ROOT$_0$],	[hit$_2$... bat$_7$],	$A_1 = \{(\text{hit}_2, \text{NSUBJ}, \text{John}_1)\}$)
RIGHT-ARC$_{ROOT}$	$\Longrightarrow$	([ROOT$_0$ hit$_2$],	[the$_3$... bat$_7$],	$A_2 = A_1 \cup \{(\text{ROOT}_0, \text{ROOT}, \text{hit}_2)\}$)
SHIFT	$\Longrightarrow$	([ROOT$_0$ hit$_2$ the$_3$],	[ball$_4$... bat$_7$],	A_2)
LEFT-ARC$_{DET}$	$\Longrightarrow$	([ROOT$_0$ hit$_2$],	[ball$_4$... bat$_7$],	$A_3 = A_2 \cup \{(\text{ball}_4, \text{DET}, \text{the}_3)\}$)
RIGHT-ARC$_{DOBJ}$	$\Longrightarrow$	([ROOT$_0$ hit$_2$ ball$_4$],	[with$_5$... bat$_7$],	$A_4 = A_3 \cup \{(\text{hit}_2, \text{DOBJ}, \text{ball}_4)\}$)
REDUCE	$\Longrightarrow$	([ROOT$_0$ hit$_2$],	[with$_5$... bat$_7$],	A_4)
RIGHT-ARC$_{PREP}$	$\Longrightarrow$	([ROOT$_0$ hit$_2$ with$_5$],	[the$_6$ bat$_7$],	$A_5 = A_4 \cup \{(\text{hit}_2, \text{PREP}, \text{with}_5)\}$)
SHIFT	$\Longrightarrow$	([ROOT$_0$ hit$_2$ with$_5$ the$_6$],	[bat$_7$],	A_5)
LEFT-ARC$_{DET}$	$\Longrightarrow$	([ROOT$_0$ hit$_2$ with$_5$],	[bat$_7$],	$A_6 = A_5 \cup \{(\text{bat}_7, \text{DET}, \text{the}_6)\}$)
RIGHT-ARC$_{POBJ}$	$\Longrightarrow$	([ROOT$_0$ hit$_2$ with$_5$ bat$_7$],	[],	$A_7 = A_6 \cup \{(\text{with}_5, \text{POBJ}, \text{hit}_2)\}$)

基于转移的依存分析

- arc-standard算法与arc-eager算法的区别
  - arc-standard算法会延迟进行RIGHT-ARC动作，要等待从属词的所有从属词均已完成分析后才进行RIGHT-ARC动作。
  - arc-eager算法会尽早进行RIGHT-ARC动作，不等待从属词完成分析就会进行RIGHT-ARC动作。
- 基于转移的依存分析本质上是非确定性的。
  - 每个分析格局下，有多种动作可同时进行。
- 在每个分析格局下，若能够确定可以进行的最佳动作，则算法变成确定性的分析算法。
- 确定性分析算法时间复杂度为$O(n)$

基于转移的依存分析

- 在每个分析格局下，引入一个分类器，由分类器给出当前格局下的最佳分析动作。

$$\hat{t} = \underset{t \in \mathcal{A}(c)}{\operatorname{argmax}}\, s(c,t)$$

c是分析格局，$\mathcal{A}(c)$是当前分析格局下所有可能的分析动作，$s(c,t)$是分析动作t的得分。

- 预测最佳分析动作，通常采用线性分类器

$$s(c,t) = \mathbf{w} \cdot \mathbf{f}(c,t)$$

针对分析格局提取特征，形成特征向量$\mathbf{f}(c,t)$

特征选择和参数学习

- 计算动作t的得分，需要针对分析格局提取特征
  $\beta[0] = w$ //缓冲区中第一个词是w
  $\beta[1] = noun$ //缓冲区中第二个词是名词
  $\sigma[0] = w$ //分析栈中第一个词是w
  $ld(\sigma[0]) = w$ //分析栈中的一个词的最左依存词是w
  …
- 权重向量$\mathbf{w}$是模型参数，需通过依存树库训练得到。
  - 需要将树库转换成分析动作转移序列，该转移序列称为Oracle转移序列
  - 可采用任何分类器并进行相应的参数学习，如SVM、Perceptron等

依存分析算法比较

- 基于转移的依存分析采用greedy search策略，有错误积累问题
- 基于转移的依存分析具有高效的优势($O(n)$)

- Eisner算法采用Global search策略，没有错误积累问题
- Eisner算法效率低于基于转移的依存分析($O(n^3)$)

- 两者均不能处理非投影依存结构
  - 基于图的依存分析：Chu-Liu-Edmond算法
  - 基于转移的依存分析：list-based分析算法

依存分析的评价指标

- 依存分析实质是为句子中的每个词标注(attach)中心词。
- UAS: Unlabeled Attachment Score*
  percentage of words that have the correct head.
  (只评价结构，不评价依存关系标记)
- LAS: Labeled Attachment Score *
  The percent of words that have the correct heads and labels.
  (评价结构和标记)
- RA: Root Accuracy
  percentage of sentences that have the correct root(the real root!)
- CM: Complete Match rate
  percentage of sentences in which the resulting tree was completely correct.

---

*通常不考虑标点符号。