



北京航空航天大學  
BEIHANG UNIVERSITY

# 自然语言处理

人工智能研究院

主讲教师 沙磊



# 句法分析 与 依存分析

# 句法分析导引

- 以“词”为单位的分析技术
  - 词语切分(segmentation, tokenization)
  - 形态分析(morphological analysis, lemmatization, stemming)
  - 词类标注(part-of-speech tagging)
  - .....
- 以“句”为单位的分析技术
  - 句法分析(syntactic parsing)
  - .....
- 以“篇”为单位的分析技术
  - 指代分析(anaphora resolution)
  - .....
- 句法分析关心句子的组成规律(词如何组成句子? )

# 句法学 (syntax)

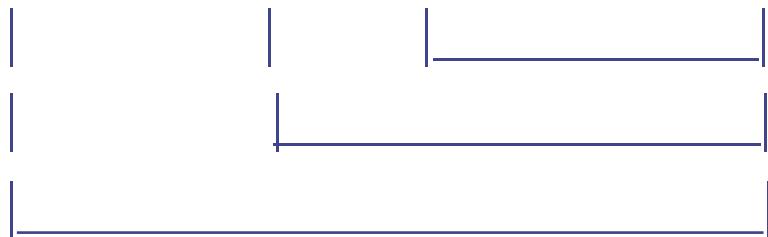
- *In linguistics, syntax is the study of the rules, or "patterned relations", that govern the way the words in a sentence come together. It concerns how different words are combined into clauses, which, in turn, are combined into sentences.*

*From Wikipedia, the free encyclopedia*

- 语言学中研究句子组成规则的学科是句法学

# 句子成分分析

- 句子是词的线性序列，但词和词之间结合的松紧程度并不一样。
  - 今天上午我们有两节课
- 句子在构造上具有层次性，较小的成分还可以进一步组成较大的成分。
  - 今天上午我们有两节课



- 不同性质的成分可以有不同的句法功能和分布，可以区分成不同的类型。
  - 名词短语(可以充当主谓结构中的主语、述宾结构中的宾语等)
    - ◆ 两节课
  - 动词短语(可以充当主谓结构中的谓语等)

# 英语中的短语举例

- 名词性短语(NP)

*the flight      the flight from Beijing to Shanghai*

*all the flights                          two flights                          a nonstop flight*

*any flights arriving after eleven a.m.*

- 动词性短语(VP)

*arrive on Sunday                          show me that book*

*buy two books                              prefer to go by Airchina*

- 介词短语(PP)

*from Beijing                              with a telescope                          from Beijing to Shanghai*

- 形容词性短语

*very easy    least expensive*

- .....

# 汉语中的短语举例

- 名词性短语(np) 两节课、机器零件
- 动词性短语(vp) 吃包子、洗干净
- 形容词性短语(ap) 很干燥、多么美妙
- 处所词性短语(sp) 地板上、盒子里
- 时间词性短语(tp) 今天上午、十年前
- 数量短语(mp) 一封(公开信)、多名(乘客)
- 介词短语(pp) 向雷锋(学习)、被老师(警告)
- .....

# 句法分析都有哪些

- 成分分析 (Constituency Parsing)
  - CFG 分析法
  - 基于统计的分析法
- 依存分析 (Dependency Parsing)
  - 规约法 (Transition-based parsing)
  - 图模型法 (Graph-based parsing)

# 句法分析都有哪些

- 成分分析 (Constituency Parsing)
  - CFG 分析法
  - 基于统计的分析法
- 依存分析 (Dependency Parsing)
  - 规约法 (Transition-based parsing)
  - 图模型法 (Graph-based parsing)

# CFG分析法----句法知识的形式化

- 上下文无关文法(CFG)是最常用的句法知识形式化工具。
- 为了便于计算机处理自然语言，计算语言学研究人员提出了许多形式语法系统(grammar formalism)，例如：功能合一语法(FUG)、词汇功能语法(LFG)、中心词驱动的短语结构语法(HPSG)等。在这些语法形式化系统中，上下文无关文法是一个核心组成部分。
- 许多句法分析算法都建立在上下文无关文法的基础上。  
(基于上下文无关文法的句法分析)

# CFG的形式定义

- 一个上下文无关文法  $G$  由四个部分组成，可记作
- $G = \{V_N, V_T, S, P\}$ , 其中：
  - $V_N$ 是非终结符号组成的有限集合
  - $V_T$ 是终结符号组成的有限集合
  - $V_N \cap V_T = \emptyset$
  - $S$ 是开始符号,  $S \in V_N$ 。
  - $P$ 是一组重写规则组成的集合, 每个重写规则具有下面的形式：
$$A \rightarrow \alpha, \text{ 其中 } A \in V_N, \alpha \in (V_N \cup V_T)^*$$

# 如何用CFG描写英语句法规则？

- 名词性短语

$NP \rightarrow Det\ Nominal$

*a flight the flight*

$NP \rightarrow ProperNoun$

*Beijing Chomsky*

$NP \rightarrow Pronoun$

*he they him*

$Nominal \rightarrow Noun \mid Noun\ Nominal$

*flight coffee bean*

$NP \rightarrow NP\ PP$

*all flights from Beijing*

.....

- 动词性短语

$VP \rightarrow Verb$

*disappear*

$VP \rightarrow Verb\ NP$

*eat a sandwich*

$VP \rightarrow Verb\ NP\ NP$

*show me the book*

$VP \rightarrow VP\ PP$

*see a girl with a telescope*

.....

# 如何用CFG描写汉语句法规则？

- 名词性短语

$$np \rightarrow n \mid r$$

$$np \rightarrow mp \ np$$

$$np \rightarrow ap \ np$$

$$np \rightarrow np \ np$$

$$np \rightarrow vp \ u$$

.....

桌子、他们

一本书、五斤牛肉

英俊的小伙子

英语教师

卖菜的

- 动词性短语

$$vp \rightarrow v$$

死

$$vp \rightarrow vp \ np$$

吃苹果

$$vp \rightarrow dp \ vp$$

认真准备

$$vp \rightarrow vp \ ap$$

洗干净

.....

# 一个简单的英语语法

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Noun Nominal$

$NP \rightarrow Proper-Noun$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$Det \rightarrow that | this | a$

$Noun \rightarrow book | flight | meal | money$

$Verb \rightarrow book | include | prefer$

$Aux \rightarrow does$

$Prep \rightarrow from | to | on$

$Proper-Noun \rightarrow Houston | TWA$

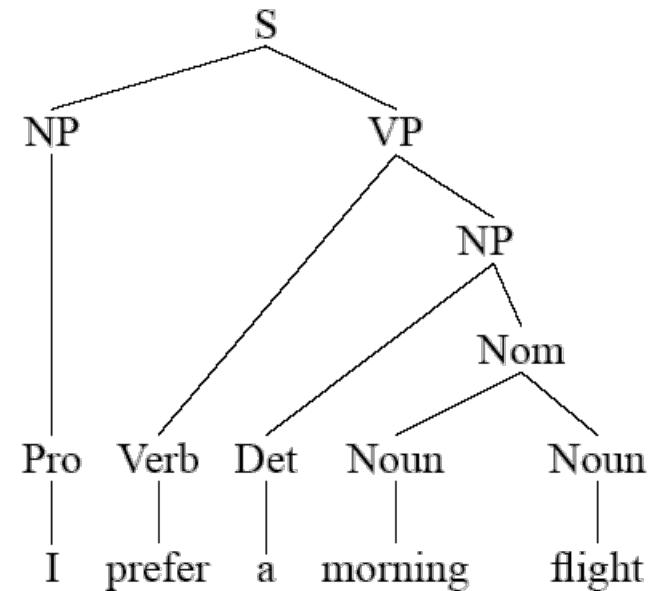
$Nominal \rightarrow Nominal PP$

# 上下文无关文法

- 作为生成装置
  - 生成语言中的句子
  - 例: *book that flight*
- 作为识别装置
  - 判别句子是否合法
  - 例: *book that flight*
- 作为分析装置
  - 产生给定句子的句法结构
  - 例: *book that flight*

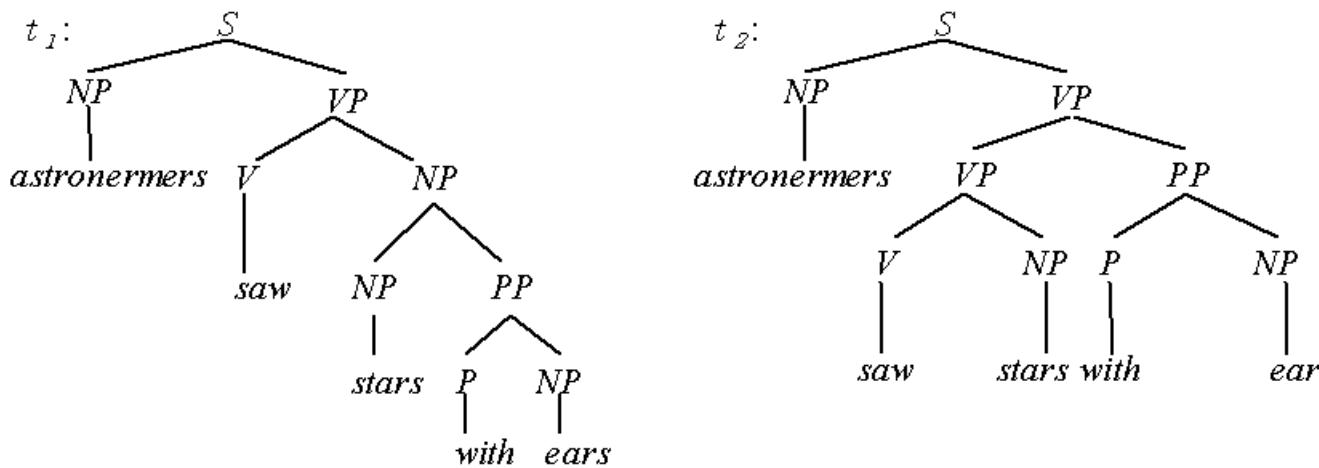
# 句法分析

- 句法分析的任务
  - 对给定的自然语言句子，分析并得到其句法结构。
  - 例如： I prefer a morning flight
- 句子的句法结构通常表示为句法树(parse)。



# 歧义

- 对于句法分析而言，所谓歧义指的是对于同一个句子，
- 按照指定的文法，会产生多种分析结果。
- 例：*astronomers saw stars with ears*



- 有时候，人理解起来没有歧义，但对计算而言，未必没有歧义。（若人理解起来有歧义，对计算机而言必有歧义）

# 自然语言的句法分析

- 通常句子越长，歧义现象越严重。以NP为例
  - NP PP PP (2)
  - NP PP PP PP (5)
  - NP PP PP PP PP(14)
- 对于有歧义的句子，句法分析器应能产生一个句子所有可能的句法分析树。
- 对于存在歧义的句子，通常在具体的上下文环境中，只有一种分析结果是正确的，句法排歧指的是根据各种知识，选择正确分析结果的过程。
- 理论上，自然语言的句法分析过程
  - (1) 利用句法分析算法生成句子所有可能的句法分析树
  - (2) 句法排歧

$$C(n) = \frac{1}{n+1} \binom{2n}{n}$$

# 人工语言的句法分析

- 为了某种交流目标而经由人工定义的语言。例如：各种程序设计语言（C语言）。
- 人工语言的特点是无二义性（或没有歧义）。
- 对于满足某种条件的人工语言，存在快速有效的语法分析方法。（这些算法大多都基于上下文无关文法）
  - LL分析法
  - LR分析法
- 甚至存在语法分析器的自动生成工具，例如Yacc。
  - 给定语言的文法，自动产生相应的语法分析程序
- 语法分析 or 句法分析
- 由于句法歧义，成熟的用于分析人工语言的句法分析算法不能直接用于自然语言的句法分析。

# 句法分析算法

- 尽管与人工语言语法分析不同，但自然语言句法分析算法大多也是基于上下文无关文法进行，即采用基于上下文无关文法的句法分析算法。
  - 自顶向下的句法分析
    - 始于开始符号  $S$
    - 利用重写规则进行推导
    - 直到推导出待分析的句子
  - 自底向上的句法分析
    - 始于待分析的句子
    - 逆向利用规则进行归约
    - 直到归约出开始符号  $S$
- [1]  $s \rightarrow np\ vp$   
[2]  $np \rightarrow n$   
[3]  $vp \rightarrow v'$   
[4]  $vp \rightarrow v' np$   
[5]  $v' \rightarrow v\ u$   
[6]  $n \rightarrow \text{曹操}$   
[7]  $v \rightarrow \text{打败}$   
[8]  $u \rightarrow \text{了}$   
[9]  $n \rightarrow \text{周瑜}$

# 自顶向下

- 采用自顶向下分析的方法分析“周瑜打败了曹操”

(1)  $s$

(2)  $s \Rightarrow np vp$  (使用[1]号规则)

(3)  $s \Rightarrow np vp \Rightarrow n vp$  (使用[2]号规则)

(4)  $s \Rightarrow np vp \Rightarrow n vp \Rightarrow$  周瑜  $vp$  (使用[9]号规则)

(5)  $s \Rightarrow np vp \Rightarrow n vp \Rightarrow$  周瑜  $vp \Rightarrow$  周瑜  $v' np$  (使用[4]号规则)

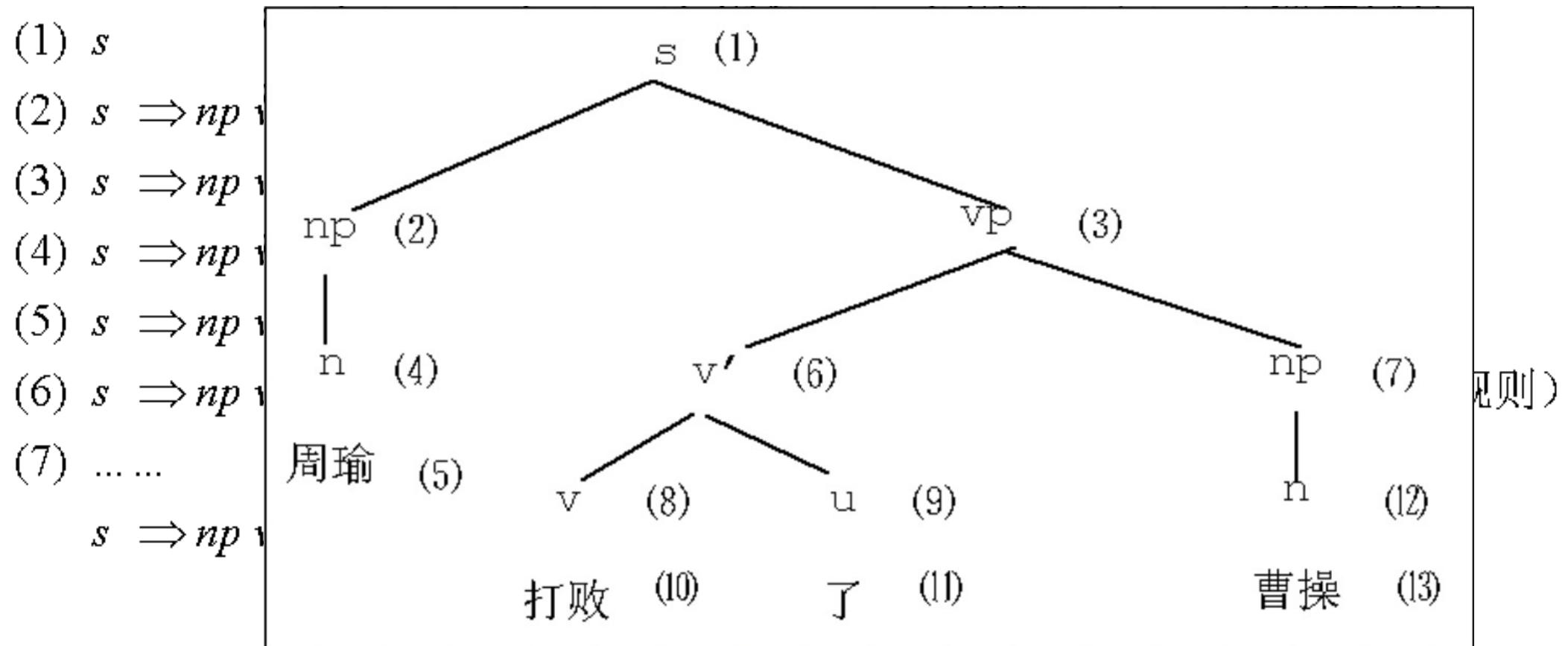
(6)  $s \Rightarrow np vp \Rightarrow n vp \Rightarrow$  周瑜  $vp \Rightarrow$  周瑜  $v' np \Rightarrow$  周瑜  $v u np$  (使用[5]号规则)

(7) ....

$s \Rightarrow np vp \Rightarrow n vp \Rightarrow \dots \dots$  周瑜 打败 了 曹操

# 自顶向下

- 采用自顶向下分析的方法分析“周瑜打败了曹操”



# 自底向上

- 采用自底向上分析的方法分析“周瑜打败了曹操”

(1) 周瑜 打败 了 曹操  $\Leftarrow n$  打败 了 曹操 (使用[9]号规则归约)

(2)  $n$  打败 了 曹操  $\Leftarrow np$  打败 了 曹操 (使用[2]号规则归约)

(3)  $np$  打败 了 曹操  $\Leftarrow np v$  了 曹操 (使用[7]号规则)

(4)  $np v$  了 曹操  $\Leftarrow np vu$  曹操 (使用[8]号规则)

(5)  $np vu$  曹操  $\Leftarrow np v'$  曹操 (使用[5]号规则)

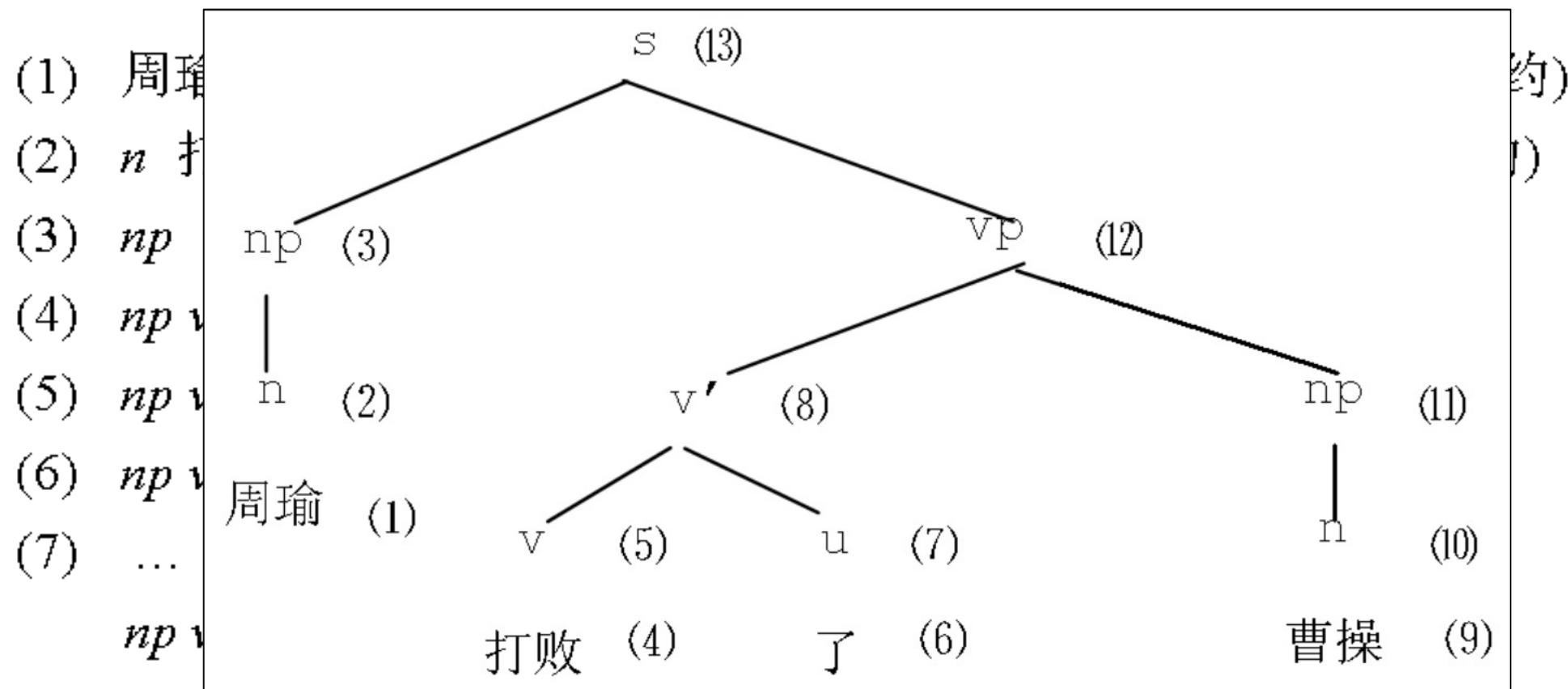
(6)  $np v'$  曹操  $\Leftarrow np v' n$  (使用[6]号规则)

(7) ...

$np vp \Leftarrow s$  (使用[1]号规则)

# 自底向上

- 采用自底向上分析的方法分析“周瑜打败了曹操”



# 分析算法的确定性

- 算法的确定性和非确定性
  - 回溯和伪并行
- 对于自顶向下的分析方法而言，当有多个重写规则可用时，如何避免选择导致分析失败的规则？
- 对于自底而上的分析方法而言，如果有多种可能的归约，如何避免导致分析失败的归约？
- 回溯导致效率低下
- 回溯也会导致重复计算
- 对于人工语言，由于语法受限，通常可以做到彻底消除非确定性，从而形成确定性分析算法。但对于自然语言而言，彻底消除非确定性是困难的。

# 分析算法的确定性

- 回溯导致的重复分析。

例如，用自顶向下的分析方法分析下面的短语

*a flight from Indianapolis to Houston on TWA*

*a flight* (4)

*from Indianapolis* (3)

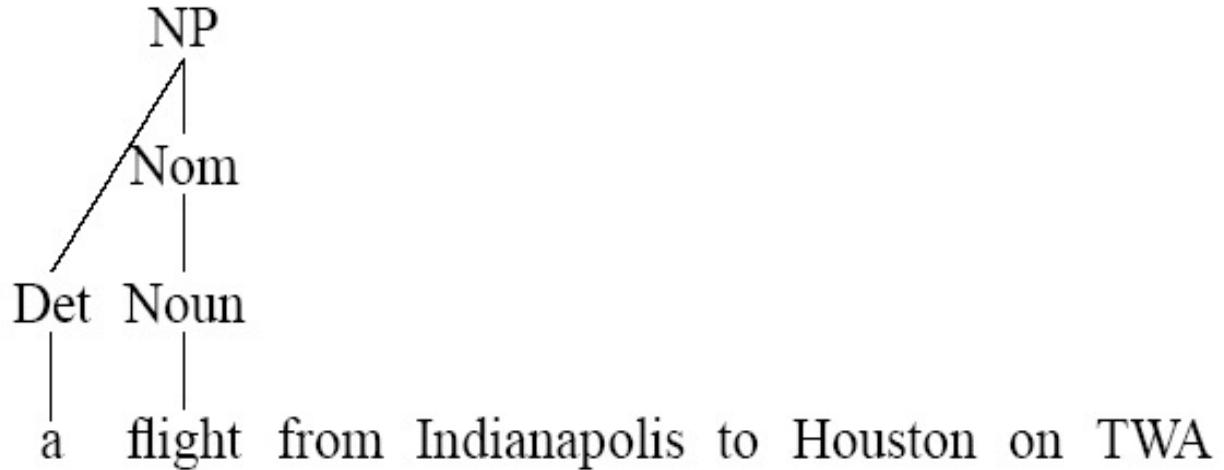
*to Houston* (2)

*on TWA* (1)

*a flight from Indianapolis* (3)

*a flight from Indianapolis to Houston* (2)

*a flight from Indianapolis to Houston on TWA* (1)



# 分析算法的确定性

- 回溯导致的重复分析。

例如，用自顶向下的分析方法分析下面的短语

*a flight from Indianapolis to Houston on TWA*

*a flight* (4)

*from Indianapolis* (3)

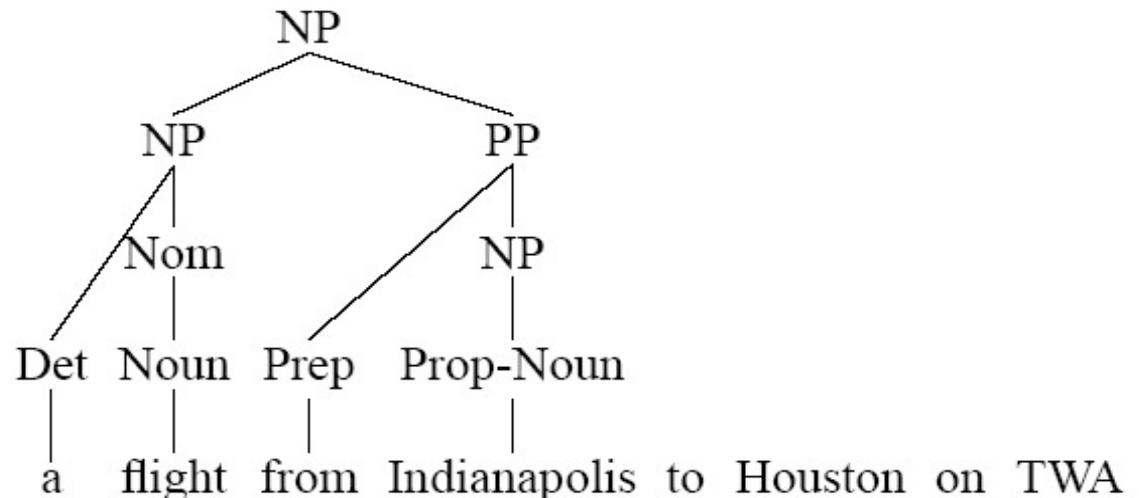
*to Houston* (2)

*on TWA* (1)

*a flight from Indianapolis* (3)

*a flight from Indianapolis to Houston* (2)

*a flight from Indianapolis to Houston on TWA* (1)



# 分析算法的确定性

- 回溯导致的重复分析。

例如，用自顶向下的分析方法分析下面的短语

*a flight from Indianapolis to Houston on TWA*

*a flight* (4)

*from Indianapolis* (3)

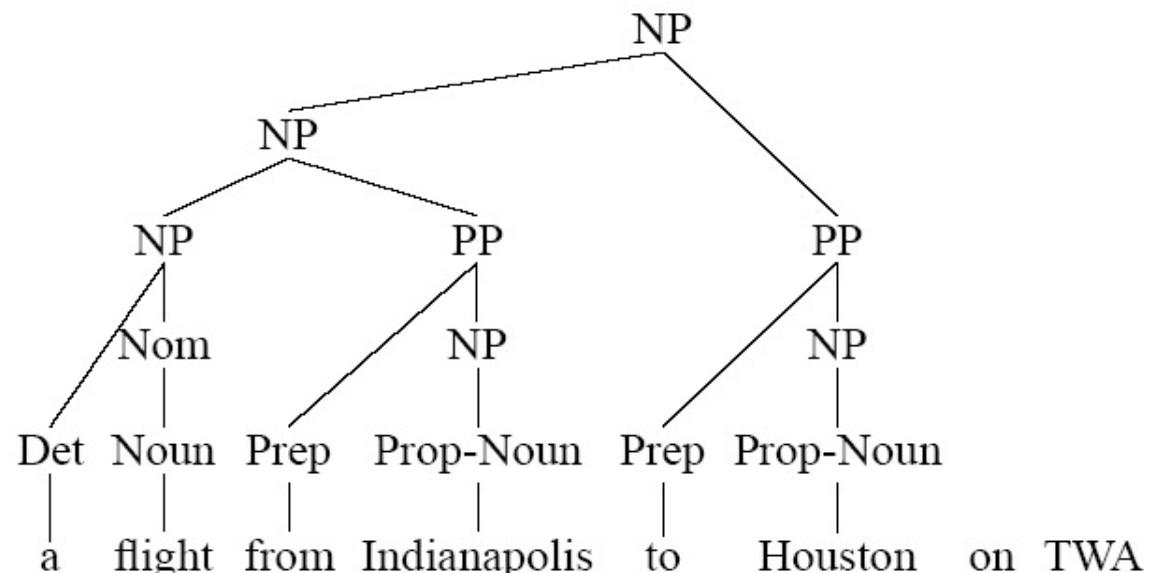
*to Houston* (2)

*on TWA* (1)

*a flight from Indianapolis* (3)

*a flight from Indianapolis to Houston* (2)

*a flight from Indianapolis to Houston on TWA* (1)



# 分析算法的确定性

- 回溯导致的重复分析。

例如，用自顶向下的分析方法分析下面的短语

*a flight from Indianapolis to Houston on TWA*

*a flight* (4)

*from Indianapolis* (3)

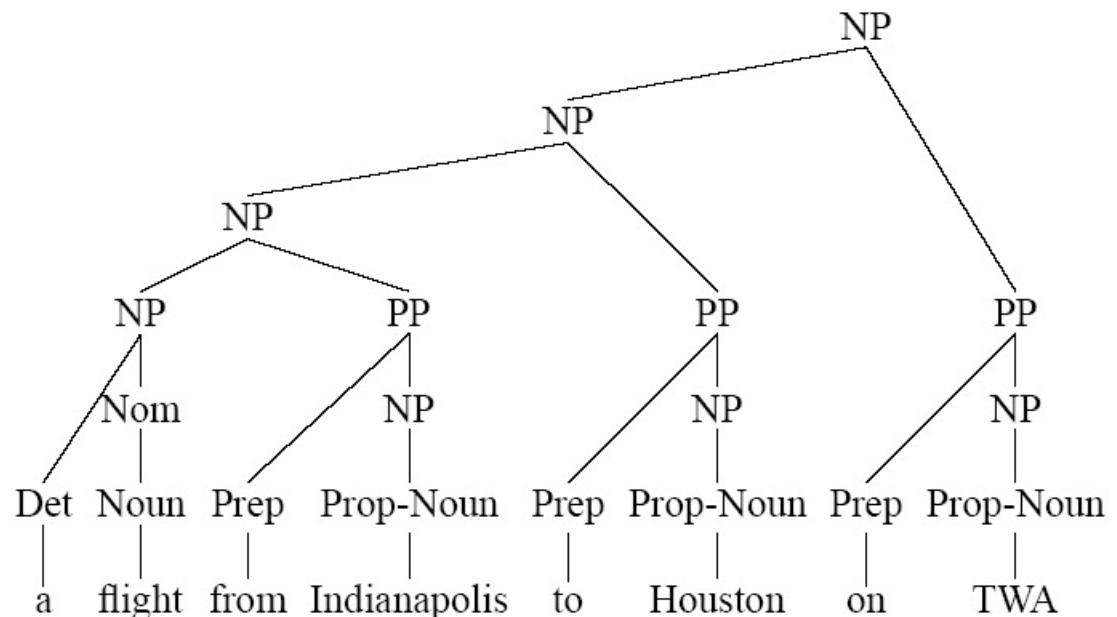
*to Houston* (2)

*on TWA* (1)

*a flight from Indianapolis* (3)

*a flight from Indianapolis to Houston* (2)

*a flight from Indianapolis to Houston on TWA* (1)



# 分析算法的确定性

- 回溯导致的重复分析。

例如，用自顶向下的分析方法分析下面的短语

*a flight from Indianapolis to Houston on TWA*

*a flight* (4)

*from Indianapolis* (3)

*to Houston* (2)

*on TWA* (1)

*a flight from Indianapolis* (3)

*a flight from Indianapolis to Houston* (2)

*a flight from Indianapolis to Houston on TWA* (1)

# 句法分析算法

- 面向自然语言，已经提出了多种句法分析算法，这些
- 算法都可以相对有效地完成句法分析。
  - Earley分析算法
  - 广义LR分析算法
  - 基于chart的分析算法
  - CYK分析算法
  - .....
- 自然语言的句法分析算法主流是非确定性的。如何提升非确定性分析算法的分析效率？
- 曾经有人提出确定性算法，很少采用(Marcus算法)。

# Earley 算法

- 1970年，由Earley提出，所以称为Earley算法。
- 属于一种自顶向下的分析算法，时间复杂度是 $O(N^3)$ ，其中 $N$ 是待分析句子中的词数。
- 主要数据结构是线图(chart)。
  - 由 $N+1$ 个状态表组成， $N$ 是句中的词数。
  - 每一个状态表对应句中的一个位置。
  - 每一个状态包含下述信息：
    - ◆ 一个子树（对应一个重写规则）。
    - ◆ 该子树的完成状态。
    - ◆ 子树与输入句子的对应关系。
- 点规则(dotted rule)：一种规则右部加点标记的重写规则。
- 如： $NP \rightarrow Det\ Nominal$

# Earley 算法

- 每一个状态对应图中一条边(弧)。如：

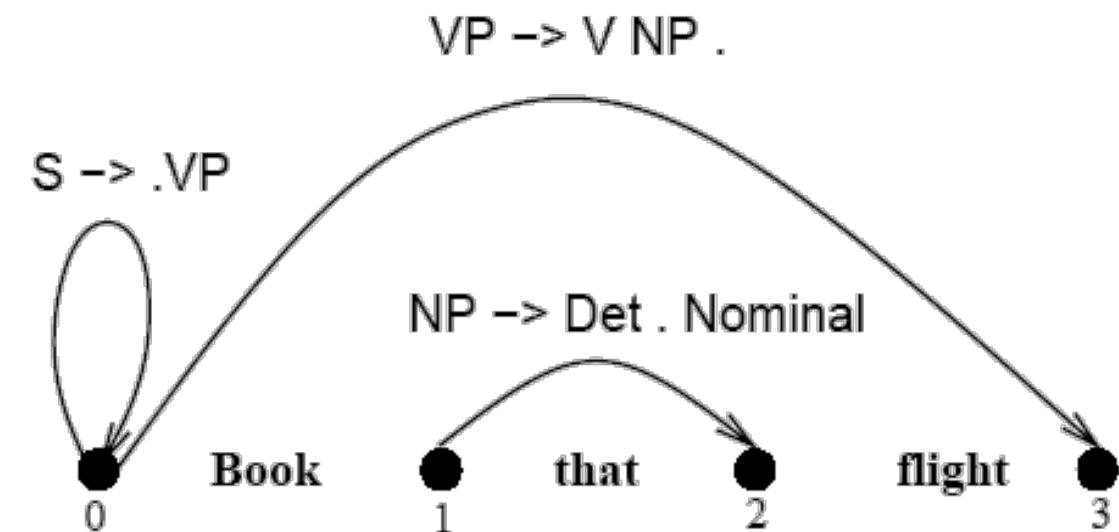
$S \rightarrow \cdot VP, [0,0]$

$NP \rightarrow Det \cdot Nominal, [1,2]$

$VP \rightarrow V NP \cdot, [0,3]$

- 图中的每个位置对应一个状态表，分别记作

$chart[0], chart[1], \dots, chart[N]$



# Earley 算法

- 三种基本操作
  - PREDICTOR(), 作用于一个点号后是非终结符号(不包括词类标记)的状态，并产生新的状态。例如：  
对于  $chart[0]$  中状态  $S \rightarrow \cdot VP, [0,0]$ ，应用 PREDICTOR() 在  $chart[0]$  中产生下列新状态  
 $VP \rightarrow \cdot Verb, [0,0]$   
 $VP \rightarrow \cdot Verb NP, [0,0]$
  - SCANNER(), 作用于一个点号后是词类标记的状态，并产生新状态。例如：  
对于  $chart[0]$  中的状态  $VP \rightarrow \cdot Verb NP, [0,0]$ ，若 PARTS-OF-SPEECH( $word[0]$ ) 为 *Verb*，应用 SCANNER()，在  $chart[1]$  中产生下面的新状态  
 $VP \rightarrow Verb \cdot NP, [0,1]$

# Earley 算法

- COMPLETER(), 作用于一个点号位于规则右部最右端的状态，并产生新状态。例如：

- 对于chart[3]中的状态 $NP \rightarrow Det\ Nominal\cdot, [1,3]$ , 应用COMPLETER(), 此时, 如果chart[1]中存在状态 $VP \rightarrow Verb\cdot NP, [0,1]$ , 则在chart[3]中产生新状态 $VP \rightarrow Verb\ NP\cdot, [0,3]$ 。
- 初始化, 在chart[0]加入一个初始状态 $\gamma \rightarrow \cdot S, [0,0]$ 。
- 终止条件, 若在chart[N]出现状态 $S \rightarrow \alpha\cdot, [0, N]$ , 分析成功。

# Earley 算法

```
function EARLEY-PARSE(words, grammar) returns chart
    ENQUEUE(( $\gamma \rightarrow \bullet S, [0, 0]$ ), chart[0])
    for i  $\leftarrow$  from 0 to LENGTH(words) do
        for each state in chart[i] do
            if INCOMPLETE?(state) and
                NEXT-CAT(state) is not a part of speech then
                    PREDICTOR(state)
                elseif INCOMPLETE?(state) and
                    NEXT-CAT(state) is a part of speech then
                    SCANNER(state)
                else
                    COMPLETER(state)
            end
        end
        return(chart)
    procedure PREDICTOR(( $A \rightarrow \alpha \bullet B \beta, [i, j]$ ))
        for each ( $B \rightarrow \gamma$ ) in GRAMMAR-RULES-FOR(B, grammar) do
            ENQUEUE(( $B \rightarrow \bullet \gamma, [j, j]$ ), chart[j])
        end
    procedure SCANNER(( $A \rightarrow \alpha \bullet B \beta, [i, j]$ ))
        if B  $\subset$  PARTS-OF-SPEECH(word[j]) then
            ENQUEUE(( $B \rightarrow \text{word}[j], [j, j+1]$ ), chart[j+1])
    procedure COMPLETER(( $B \rightarrow \gamma \bullet, [j, k]$ ))
        for each ( $A \rightarrow \alpha \bullet B \beta, [i, j]$ ) in chart[j] do
            ENQUEUE(( $A \rightarrow \alpha B \bullet \beta, [i, k]$ ), chart[k])
        end
    procedure ENQUEUE(state, chart-entry)
        if state is not already in chart-entry then
            PUSH(state, chart-entry)
        end
```

# Earley 算法

- 分析实例，例如分析句子 *book that flight*

$word[0] = "book"$

$PARTS-OF-SPEECH(word[0]) = \{Verb, Noun\}$

$chart[0]$

$\gamma \rightarrow \cdot S, [0,0] PREDICTOR()$

$S \rightarrow \cdot NP VP, [0,0] PREDICTOR()$

$S \rightarrow \cdot Aux NP VP, [0,0], SCANNER()$

$S \rightarrow \cdot VP, [0,0] PREDICTOR()$

$NP \rightarrow \cdot Det Nominal, [0,0], SCANNER()$

$NP \rightarrow \cdot Proper-Noun, [0,0], SCANNER()$

$VP \rightarrow \cdot Verb, [0,0], SCANNER()$

$VP \rightarrow \cdot Verb NP, [0,0], SCANNER()$

# Earley 算法

- 分析实例，例如分析句子 *book that flight*

$word[0] = "book"$

$PARTS-OF-SPEECH(word[0]) = \{Verb, Noun\}$

$word[1] = "that"$

$PARTS-OF-SPEECH(word[1]) = \{Det\}$

$chart[1]$

$Verb \rightarrow book \cdot, [0,1], COMPLETER()$

$VP \rightarrow Verb \cdot, [0,1], COMPLETER()$

$VP \rightarrow Verb \cdot NP, [0,1], PREDICTOR()$

$S \rightarrow VP \cdot, [0,1], COMPLETER()$

$NP \rightarrow Det Nominal, [1,1], SCANNER()$

$NP \rightarrow Proper-Noun, [1,1], SCANNER()$

# Earley 算法

- 分析实例，例如分析句子 *book that flight*

$word[0] = "book"$	
$PARTS-OF-SPE$	
$word[1]=$	$word[2]= "flight"$
$PARTS-OF$	$PARTS-OF-SPEECH(word[2]) = \{Noun\}$
$chart[1]$	$chart[2]$
$Verb \rightarrow book$	$Det \rightarrow that, [1,2], COMPLETER()$
$VP \rightarrow Verb$	$NP \rightarrow Det \cdot Nominal, [1,2], PREDICTOR()$
$VP \rightarrow Verb$	$Nominal \rightarrow \cdot Noun, [2,2], SCANNER()$
$S \rightarrow VP \cdot, [0,1]$	$Nominal \rightarrow \cdot Noun \ Nominal, [2,2], SCANNER()$
$NP \rightarrow Det$	
$NP \rightarrow Proper-Noun$	$[1,1], SCANNER()$

# Earley 算法

- 分析实例，例如分析句子 *book that flight*
- ENQUEUE() 保证不会反复分析某些子树。算法是非确定性的，需要尝试各种可能。
- 如果希望产生树结构，算法仍需要修正，给状态增加相应的指针

# 标准LR算法

- 标准LR分析算法是为分析程序设计语言等人工语言而提出的，由Knuth于1965提出。
- 标准LR分析算法，是一种自底向上的分析算法。
- 标准LR分析算法效率高且应用广泛，在许多程序设计语言的编译器中得到应用。
- 并非所有语言都可以使用标准LR分析算法进行分析，只有LR文法所定义的语言可以使用LR分析算法进行分析。
- 对于LR文法所定义的语言，LR分析算法，完全消除了回溯，可以以确定性的方式进行分析。
- 标准LR分析算法是确定性分析算法

# 标准LR算法

- 利用标准LR分析算法进行分析，首先要构造LR分析表。
- 构造LR分析表首先要构造所有的分析状态和状态转移图。
- 如何构造LR分析表，可参阅《编译原理》等相关教材。
- 一个LR分析表由两个部分组成，一部分为动作表(ACTION)，另一部分为转移表(GOTO)。
- LR分析算法通过查分析表来完成分析过程。
- LR分析算法主要使用两个数据结构，分析栈以及输入缓冲区，分析栈用来记录分析过程的中间状态，输入缓冲区用来保存待分析的句子。

# 标准LR算法

- 标准LR分析算法主要有两种分析动作
  - 移入动作( $s$ ): 把输入缓冲区中输入指针指向的词移入分析栈。
  - 归约动作( $r$ ): 运用重写规则, 把分析栈顶的符号串替换成一个非终结符号。
- 对于LR分析器而言, 除了进行分析动作外, 还要考虑分析状态的转移, 通常用 $sn$ 表示移入动作, 其中 $n$ 表示移入动作发生后, 分析器所处的状态。用 $rj$ 表示归约, 表示用第 $j$ 条重写规则进行归约, 归约后分析器所处的状态可以查询转移表(GOTO)获得。
- 初始化: 分析栈中首先放一个状态0, 输入缓冲区中放待分析的句子, 输入指针指向待分析句子中第一个词。

# 标准LR算法

算法：LR 分析算法

1. 把状态 0 压入分析栈，把  $w\$$  放在输入缓冲区中；
2. 循环执行下面的语句：
  - a) 令  $s$  是分析栈的栈顶状态，并且  $a$  是输入缓冲区中第一个符号；
  - b) 若  $\text{ACTION}[s, a] = si$ , 则把  $a$  和状态  $i$  先后压入分析栈中；
  - c) 若  $\text{ACTION}[s, a] = ri$ , 并且第  $i$  条产生式为  $A \rightarrow \beta$  则, 若  $\beta$  中包含的语法符号数为  $|\beta|$ , 则：
    - i. 从栈顶弹出  $2 \times |\beta|$  个符号(因为同时要弹出状态号);
    - ii. 令  $i$  为当前栈顶状态;
    - iii. 把  $A$  和  $\text{GOTO}[i, A]$  先后推入分析栈中;
  - d) 若  $\text{ACTION}[s, a] = \text{acc}$ , 则：
    - i. 宣布分析成功;
    - ii. 算法结束;
  - e) 若  $\text{ACTION}[s, a]$  是空白, 则：
    - i. 宣布分析失败;
    - ii. 算法结束。

# 广义LR算法

- 标准LR文法不能有二义性(歧义)，因此标准LR分析算法不能用来分析自然语言。
- 对于描述自然语言的上下文无关文法可以使用同样的方法构造LR分析表，但构造出的分析表不是确定性的分析表。也就是说，动作表的一个单元格内可能包含多个分析动作，或者说分析表具有多重入口。
- 当分析表的单元格中出现多个动作时，标准LR分析器不知道应该执行哪个分析动作。
- 例如，针对下述文法，构造LR分析表

[0] $S' \rightarrow S$		
[1] $S \rightarrow NP VP$	[2] $VP \rightarrow V$	[3] $VP \rightarrow VNP$
[4] $VP \rightarrow VNP NP$	[5] $VP \rightarrow VP PP$	[6] $NP \rightarrow Det N$
[7] $NP \rightarrow Pron$	[8] $NP \rightarrow NP PP$	[9] $PP \rightarrow Prep NP$

# 广义LR算法

状态	ACTION						GOTO			
	<i>Det</i>	<i>N</i>	<i>Prep</i>	<i>Pron</i>	<i>V</i>	\$	<i>NP</i>	<i>PP</i>	<i>S</i>	<i>VP</i>
0	s2				s3		1		4	
1			s8			s6		7		5
2		s10								
3	r7		r7	r7	r7	r7				
4							acc			
5		s8					r1		9	
6	s2		r2	s3			r2	11		
7	r8		r8	r8	r8	r8				
8	s2			s3				13		
9			r5				r5			
10	r6		r6	r6	r6	r6				
11	s2		s8/r3	s3			r3	12	7	
12			s8/r4				r4		7	
13	r9		s8/r9	r9	r9	r9		7		

# 广义LR算法

- 1987年，日本学者富田胜(Tomita)对标准LR算法进行了改进，使之可以用来分析自然语言，即广义LR算法，又称富田胜算法或Tomita算法。
- 对于LR分析表中的多重入口，由于相应的分析动作是多重的，分析动作应同时沿着多条分析路径进行。富田胜为此引入了图结构栈技术。在分析过程中，每当分析进程遇到有多个动作同时可以进行，分析进程就分裂成相应的几个子进程。栈顶亦分裂为多个栈顶，分别依据分析表中规定的不同动作进行分析。如果两个进程处理同一状态，则栈顶合并为一个栈顶，两个进程则合并为一个进程，这样就形成一种图结构的分析栈。

# 广义LR算法

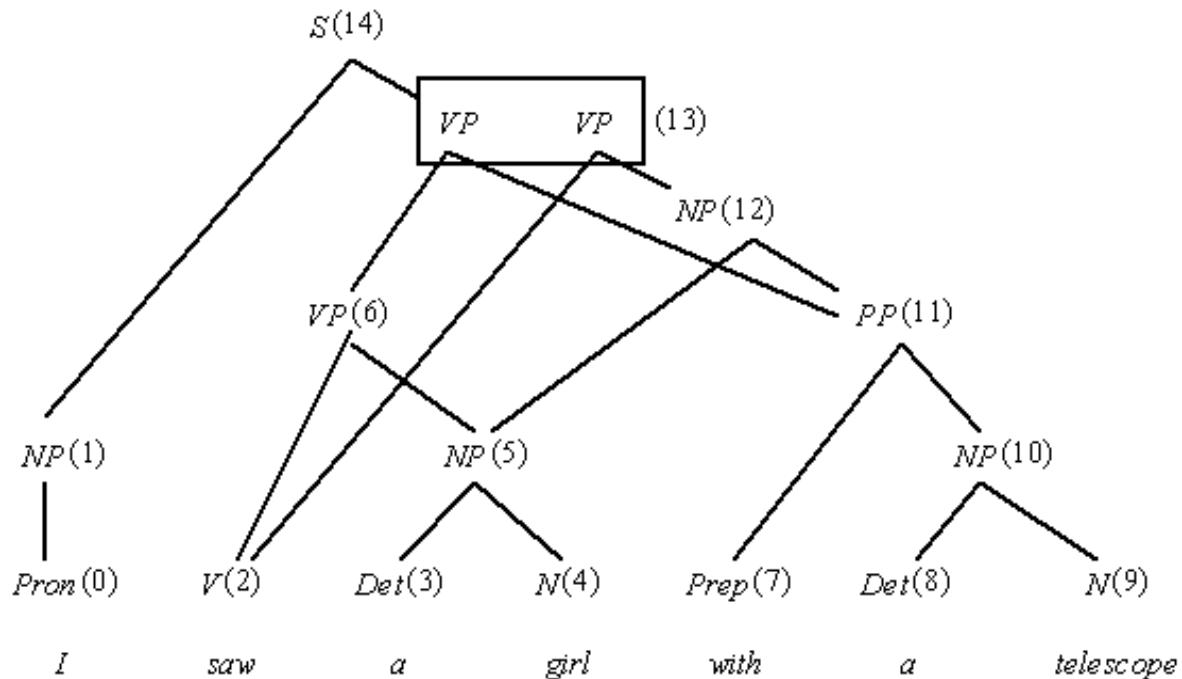
- 分析树的构造
  - 当分析器移进一个词时，就用该词和相应的词类标记创立叶子结点
  - 当分析器归约时，利用归约的规则创建相应的分支结点
- 子树共享
  - 如果两棵或两棵以上的树具有共同的子树，那么这棵子树就只应该表示(构造)一次。
  - 为了构造共享子树，分析过程不再把语法符号入栈，而是将指向子树的指针入栈。
  - 创建树结点时，使用这些指向子树的指针创建新结点。

# 广义LR算法

- 局部歧义：如果两棵或两棵以上子树的所有叶结点都相同，并且所有子树的根结点被标有同一非终结符号，也就是说句子的某一部分能用两种或两种以上方式归约为同一非终结符，这时称句子中出现了局部歧义。
- 局部歧义压缩
  - 如果句子中有许多局部歧义，总的歧义数将会指数增长。为避免这种增长，可采用了局部歧义压缩技术。这种技术是把有局部歧义的子树的结点结合为一体，这样的结点叫收集结点。
  - 在图结构栈中，如果两个或多个符号顶点左边具有一个共同的状态顶点，并且右边有一个共同的状态顶点，则表示这几个符号顶点具有局部歧义。

# 广义LR算法

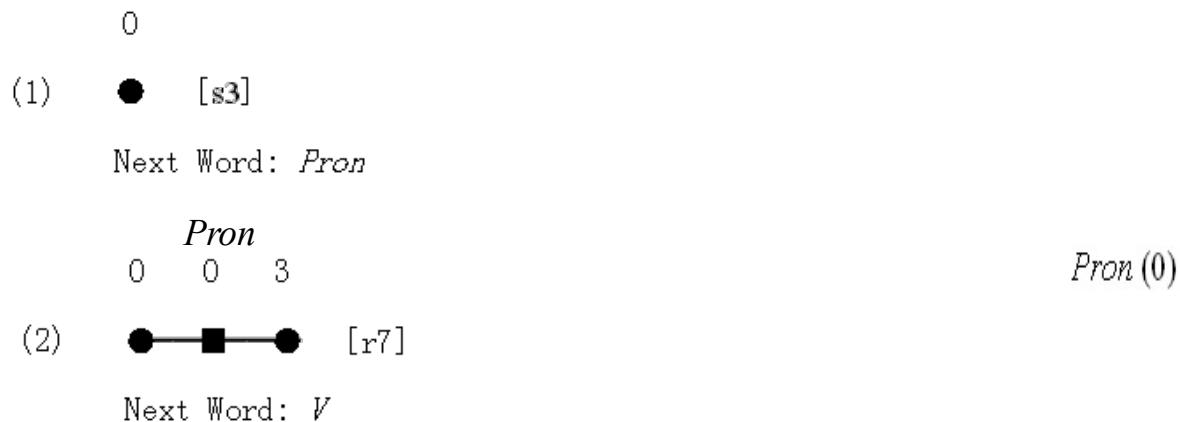
- 压缩共享森林：采用了子树共享技术和局部歧义压缩技术后，得到的分析结果被称为压缩共享森林。



- 压缩共享森林是一个句子的多个句法树的压缩表示法。

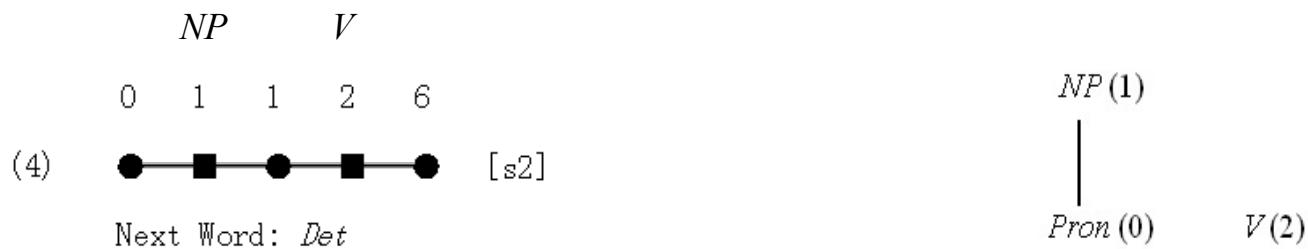
# 广义LR算法

- 广义LR分析算法除上述改进之外，其分析过程同LR分析算法基本相同。
- 分析实例： $I \quad saw \quad a \quad girl \quad with \quad a \quad telescope$   
 $Pron \quad V \quad Det \quad N \quad Prep \quad Det \quad N \quad \$$
- 下图中，●表示图结构栈中的状态，■表示压入栈中的指针

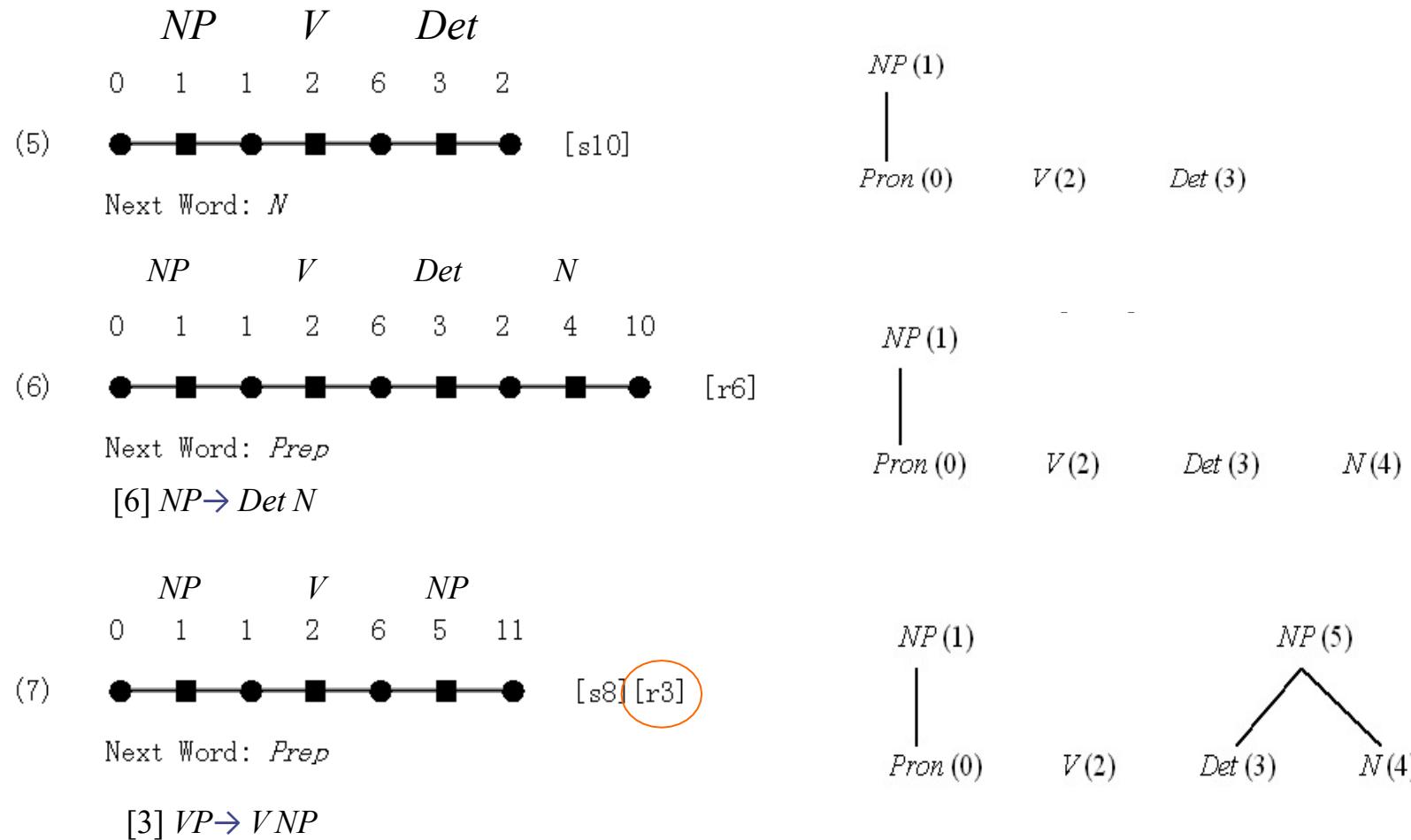


[7]  $NP \rightarrow Pron$

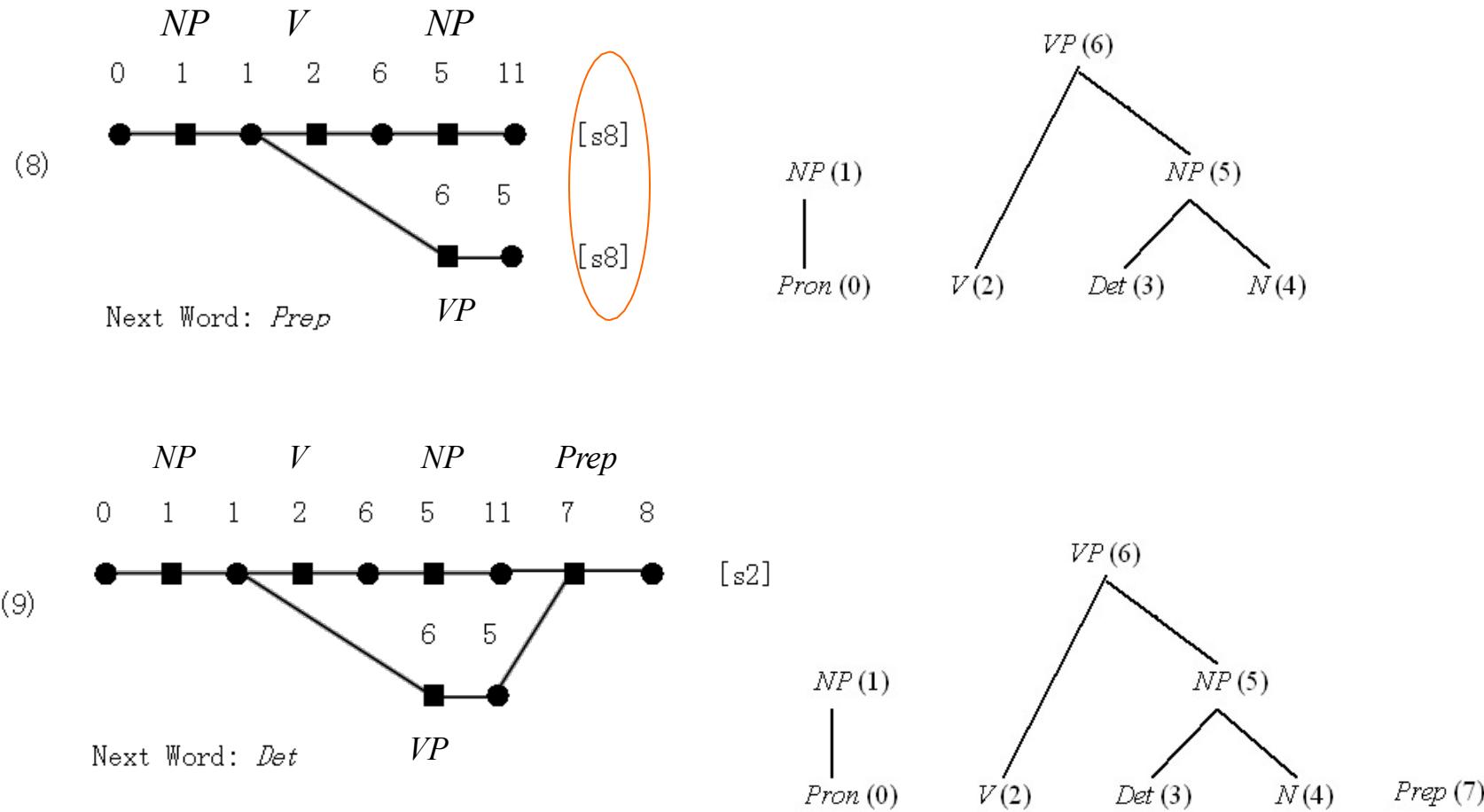
# 广义LR算法



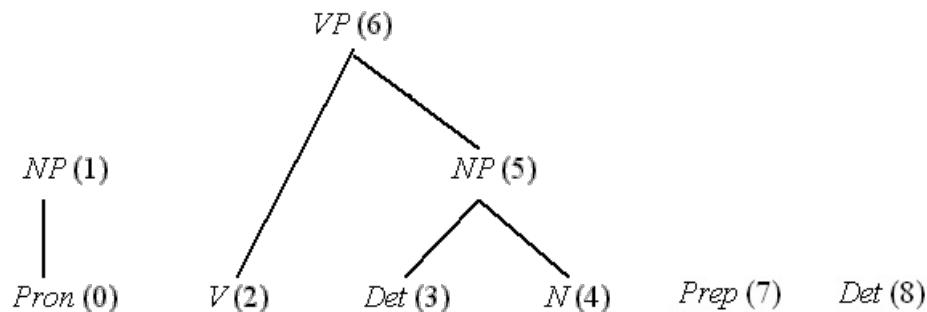
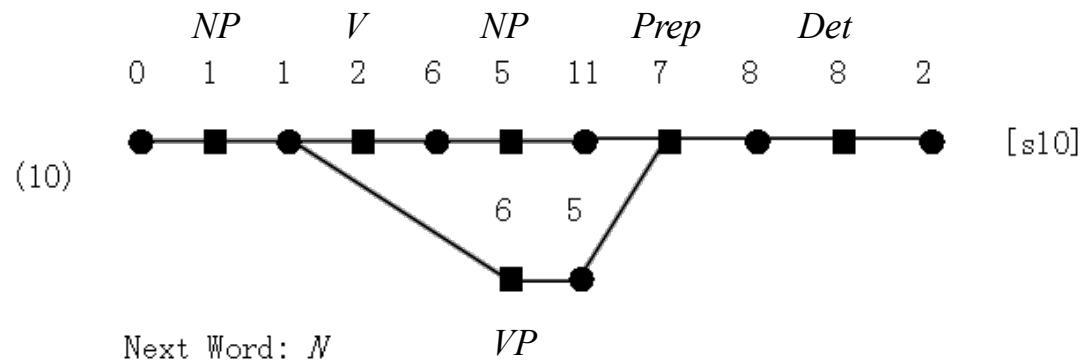
# 广义LR算法



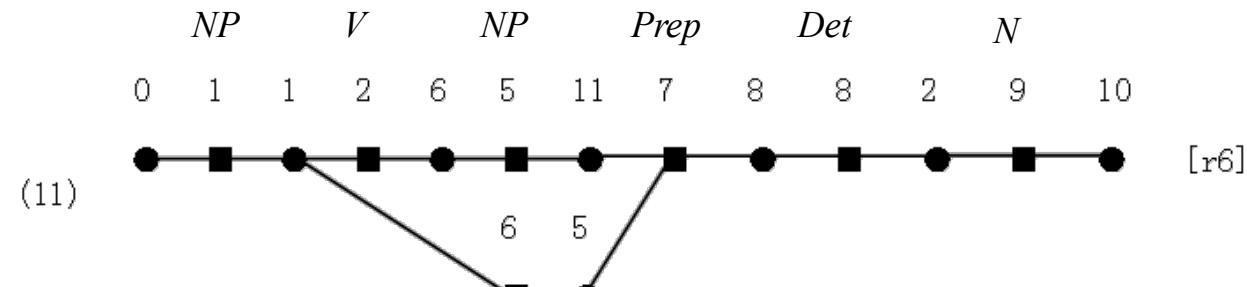
# 广义LR算法



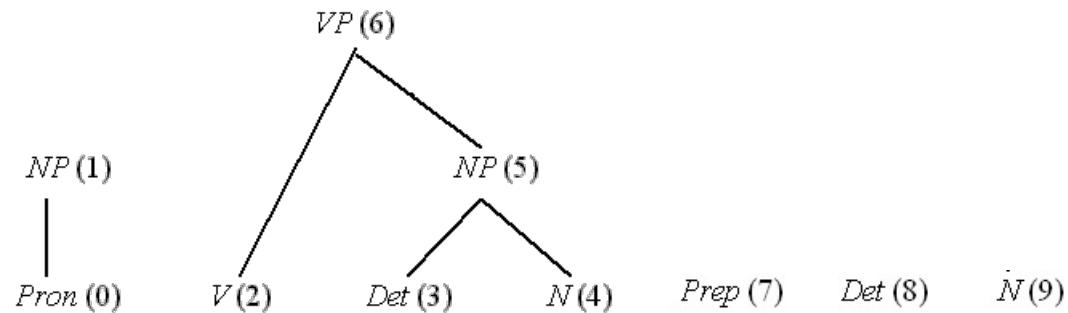
# 广义LR算法



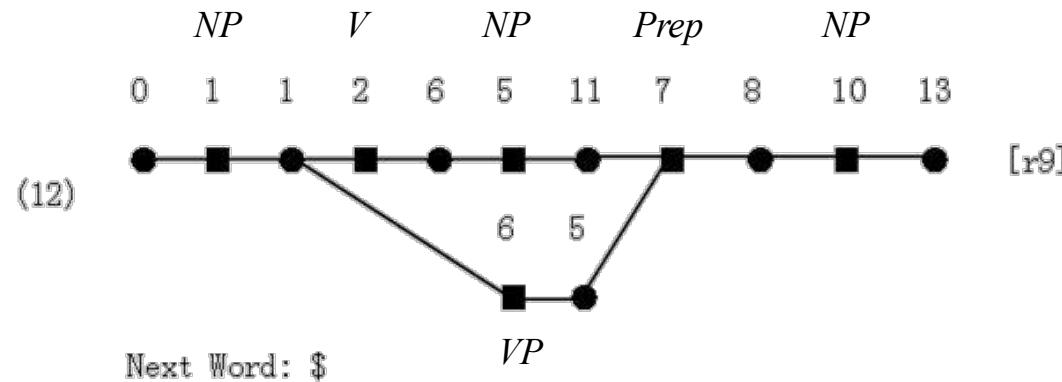
# 广义LR算法



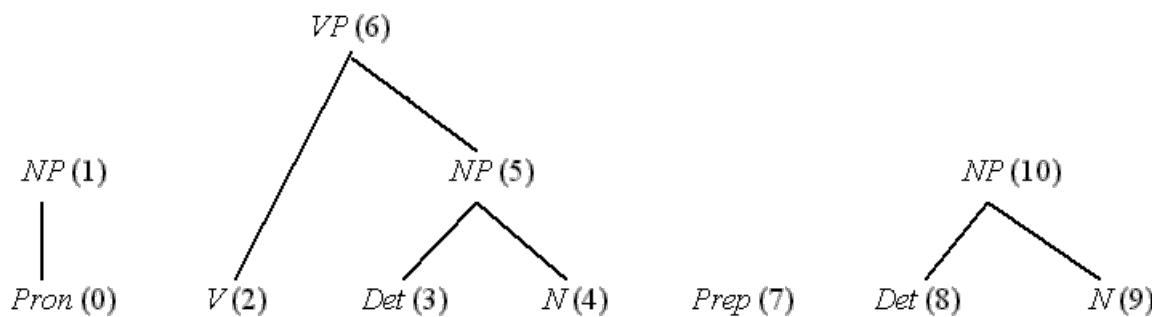
[6]  $NP \rightarrow Det\ N$



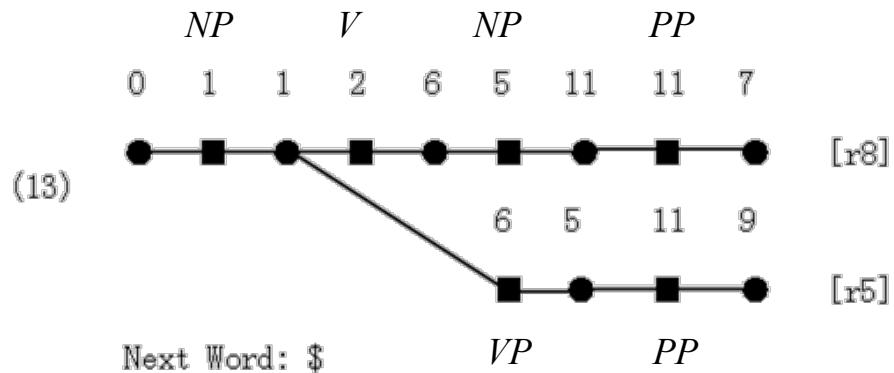
# 广义LR算法



[9]  $PP \rightarrow Prep\ NP$

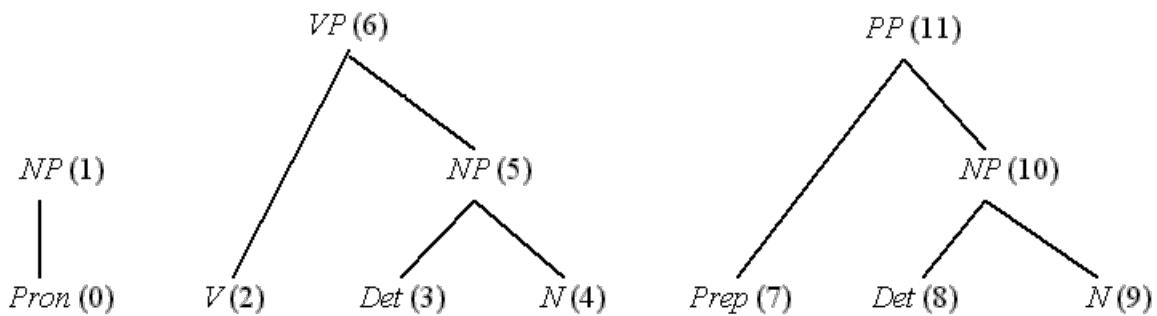


# 广义LR算法

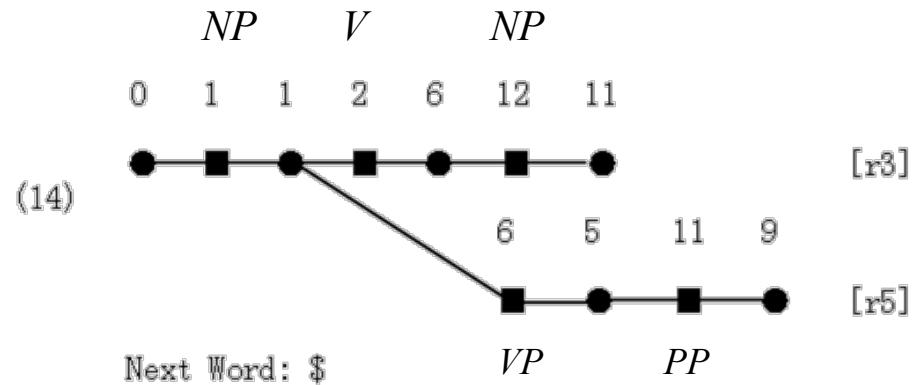


[8]  $NP \rightarrow NP\ PP$

[5]  $VP \rightarrow VP\ PP$

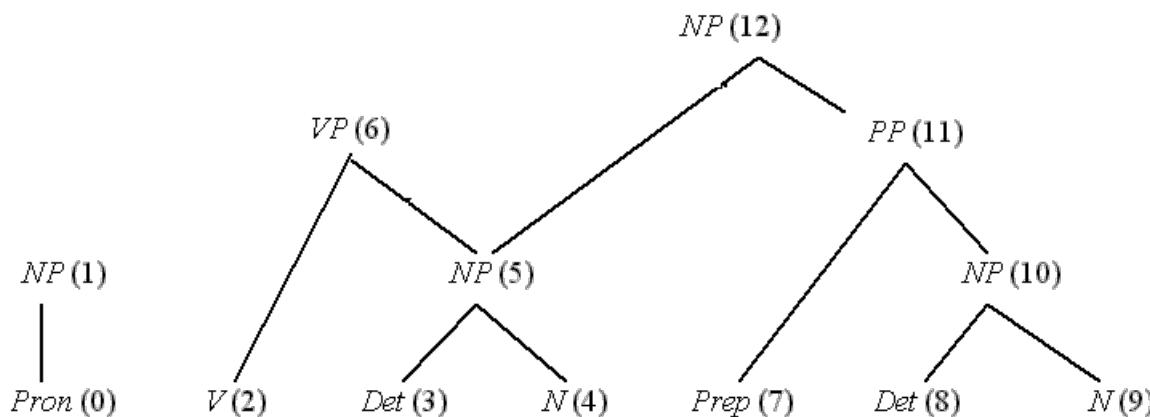


# 广义LR算法

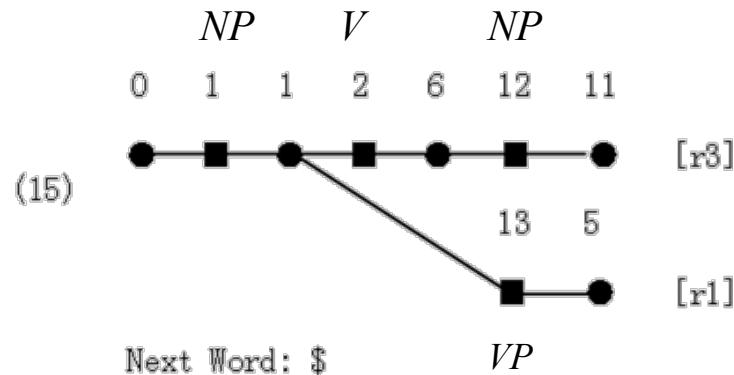


[3]  $VP \rightarrow VNP$

[5]  $VP \rightarrow VP\ PP$

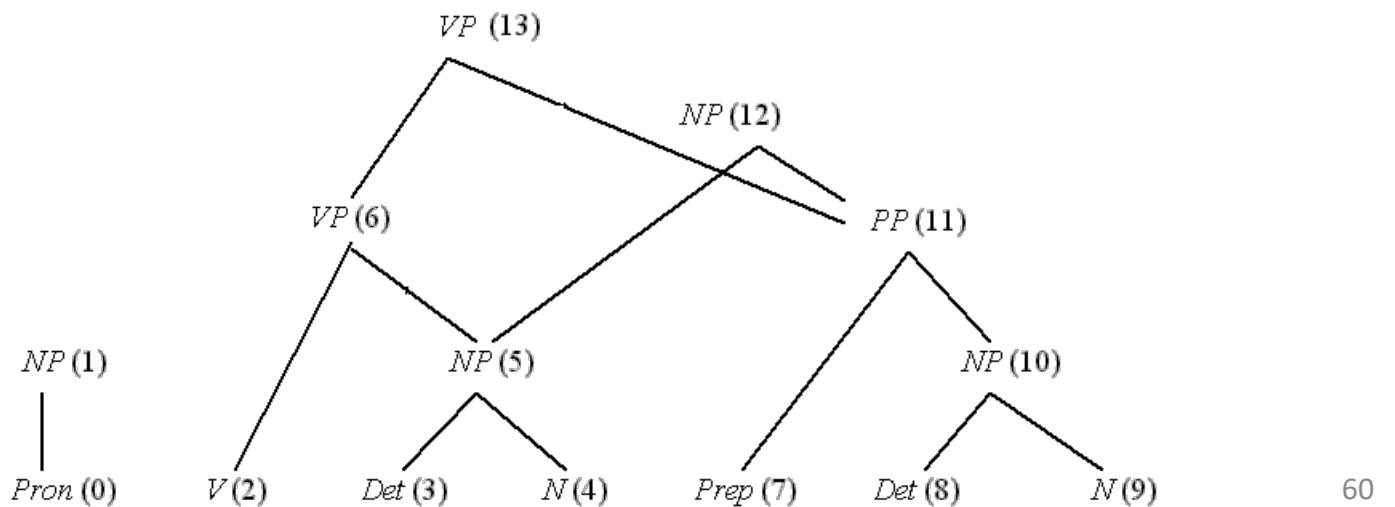


# 广义LR算法

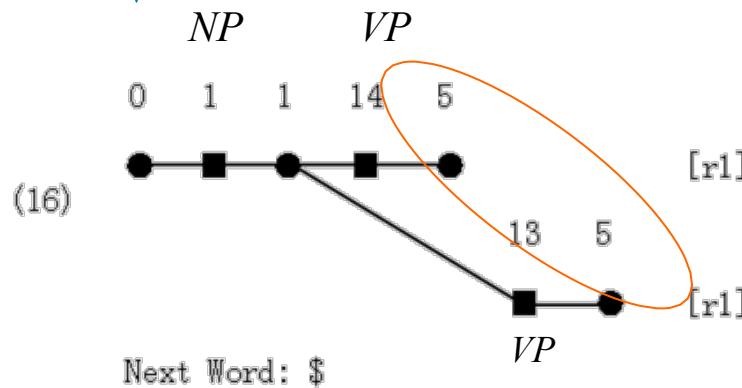


[3]  $VP \rightarrow VNP$

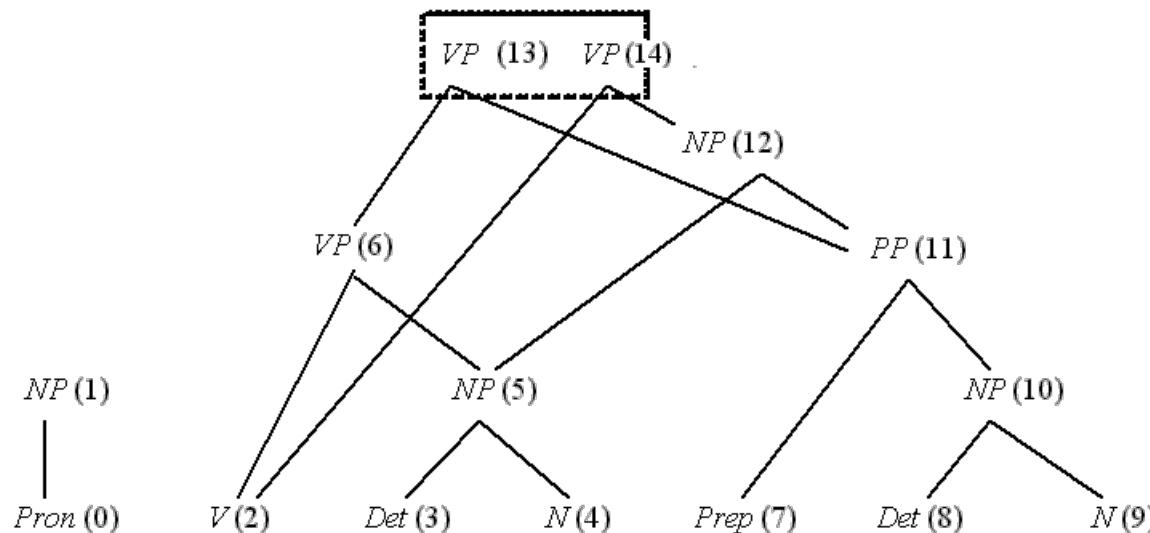
[1]  $S \rightarrow NP VP$



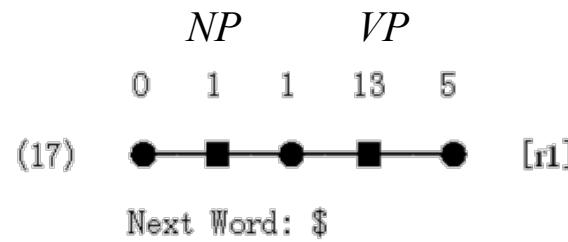
# 广义LR算法



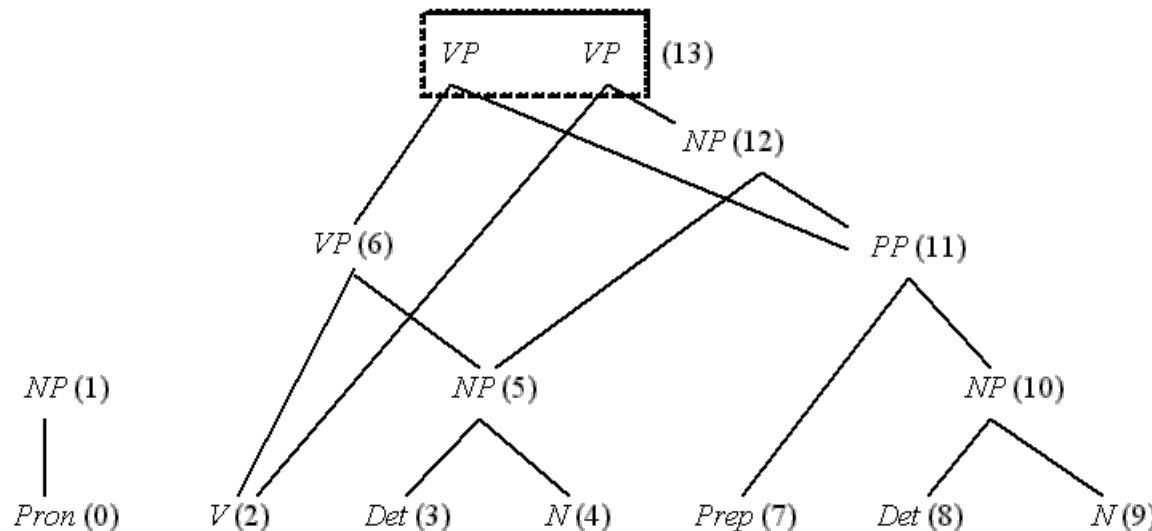
[1]  $S \rightarrow NP VP$



# 广义LR算法

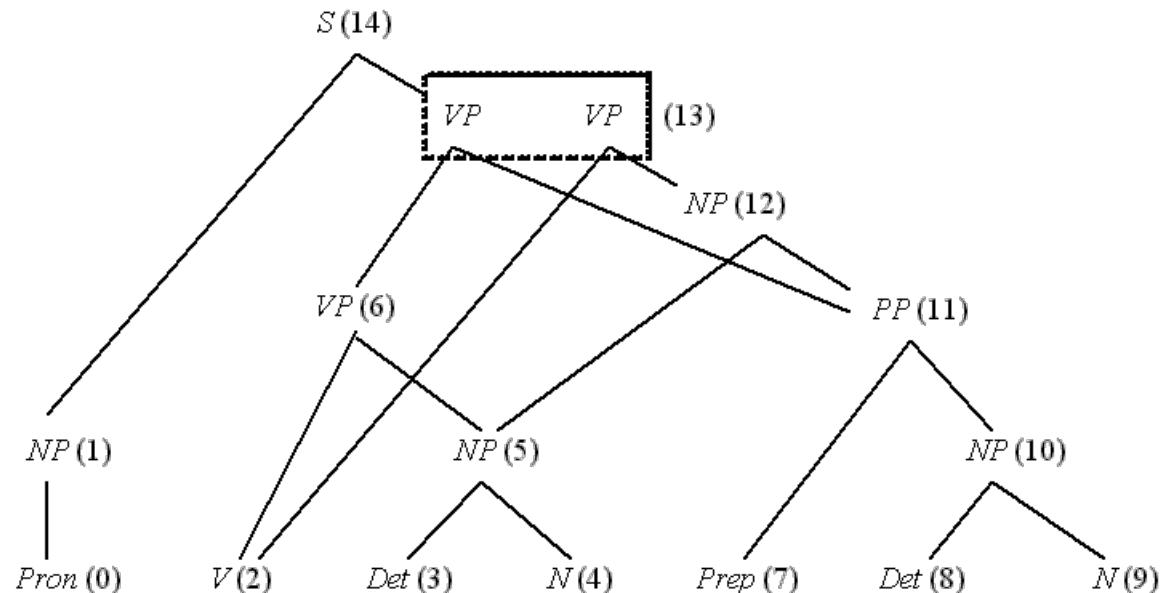


[1]  $S \rightarrow NP VP$



# 广义LR算法

*S*  
0    14    4  
(18)    ●—■—● [acc]  
Next Word: \$



# 广义LR算法总结

- 广义LR分析算法是非确定性算法
- 广义LR分析算法通过构造分析表消除了部分回溯和子树的重复分析
- 通过子树共享和局部歧义压缩实现了分析结果的高效存储---压缩共享森林
- Earley算法、GLR算法只是众多句法分析算法的代表，还有其它许多分析算法，可自行了解

# 句法分析都有哪些

- 成分分析 (Constituency Parsing)
  - CFG 分析法
  - 基于统计的分析法
- 依存分析 (Dependency Parsing)
  - 规约法 (Transition-based parsing)
  - 图模型法 (Graph-based parsing)

# 基于统计的句法分析

- 句法分析，理论上可由两个阶段完成生成句子的所有句法树  
句法排歧，找出正确的句法树
- 如何评价所有的句法树
- 引入概率，建立基于统计的句法分析
- 统计句法分析在一段时间内引起了较多的关注，并取得了较好的研究成果。

# 统计句法分析的基本思路

- 对给定的句子  $S$ , 该句子的统计句法分析结果为:

$$\hat{T} = \arg \max_T P(T|S)$$

- 根据贝叶斯公式, 有:

$$\hat{T} = \arg \max_T P(T, S)$$

- 如何计算句法树的概率?
  - 概率上下文无关文法 (PCFG)
  - 改进的 PCFG

# CKY算法

- 属于基于CFG的句法分析算法(与GLR、Earley类似)
- 由Cocke、Kasami及Younger，上世纪60年代提出，故命名为：Cocke-Kasami-Younger算法，简称CKY算法。
- 属于自底而上的分析算法。
- 可以处理一般的文法，但更适于处理乔姆斯基范式。
- 什么是乔姆斯基范式？文法中只能有下面两种形式的重写规则：
  - $A \rightarrow BC$
  - $A \rightarrow w$

# CKY算法

- 可以证明，对任何一个CFG，都存在与之等价的乔姆斯基范式。
  - $S \rightarrow aAB|BA \quad A \rightarrow BBB|a$
  - $B \rightarrow AS|b$
- 假定文法符合乔姆斯基范式不会损失CKY算法的一般性。 CKY 算法的主要数据结构是一个二维表  $T[n,n]$ ，其中每个表元素定义如下：

$$t_{i,j} = \{A \mid A \xrightarrow{+} w_i w_{i+1} \dots w_j\}$$

- 即可以推导出词串  $w_i w_{i+1} \dots w_j$  的非终结符号所组成的集合。

# CKY算法

- CKY算法自底而上填写表格，首先

$$t_{i,j} = \{A \mid A \rightarrow w_i \in P\}$$

- 若有  $A \rightarrow BC \in P$ ,  $B \in t_{i,k}$  及  $C \in t_{k+1,j}$ , 则有  $A \in t_{i,j}$ 。为什么？
- 分析过程举例，给定文法

$$S \rightarrow AA|AS|b$$

$$A \rightarrow SA|AS|a$$

试分析句子abaab。

	1	2	3	4	5
1	A	S,A	S,A	S,A	S,A
2	S	A	S	S,A	
3		A	S	S,A	
4			A	S,A	
5				S	

# CKY 算法

```
FUNCTION CKYRecognizer() begin
    boolean chart[1..n][1..|N|][1..n]←FALSE;
    for  $k \leftarrow 1$  to  $n$  do
        for each rule  $A \rightarrow w_k \in P$  do
             $chart[k, A, k] \leftarrow TRUE;$ 
    for  $l \leftarrow 2$  to  $n$  do
        for  $p \leftarrow 1$  to  $n-l+1$  do
            for  $t \leftarrow 1$  to  $l-1$  do
                 $q \leftarrow p+l-1; d \leftarrow p+t-1;$ 
                for each rule  $A \rightarrow BC \in P$  do
                     $chart[p, A, q] \leftarrow chart[p, A, q] \vee$ 
                    ( $chart[p, B, d] \wedge chart[d+1, C, q]$ );
    return chart[1, S, n];
end.
```

# 什么是概率上下文无关文法？

- PCFG是CFG的一种扩展。一个PCFG
- $G$  是一个四元组  $G = (N, \Sigma, S, P)$

其中：

$N$  有限个非终结符号组成的集合

$\Sigma$  有限个终结符号组成的集合

$S$  文法的开始符号

$P$  是一组带有概率信息的重写规则组成的集合，每条规则形式如下：

$$A \rightarrow \alpha \quad [P(A \rightarrow \alpha)]$$

$\alpha \in (V_N \cup V_T)^*$ ,  $P(A \rightarrow \alpha)$  是重写规则的概率。且：  $\sum_j P(A \rightarrow \alpha_j) = 1$

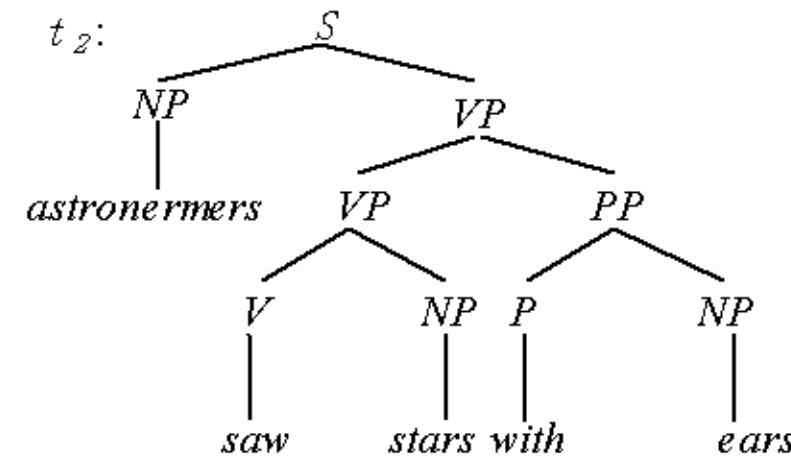
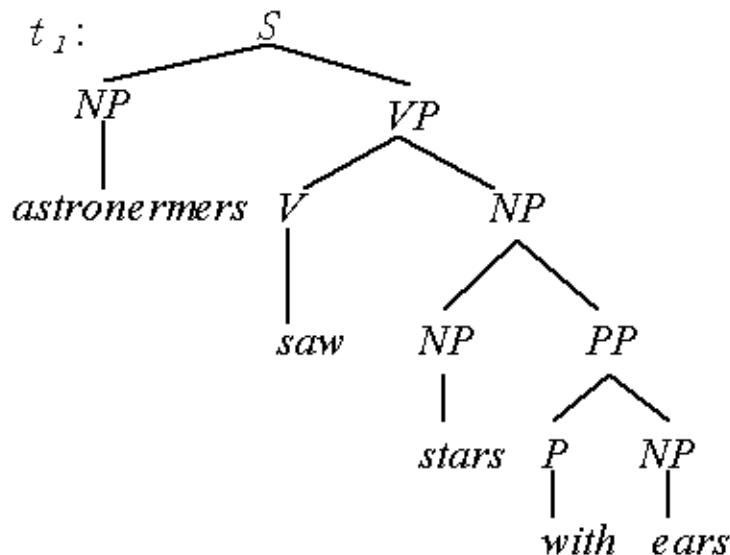
# 概率上下文无关文法举例

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow \text{astronomers}$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow ears$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow saw$	0.04
$P \rightarrow with$	1.0	$NP \rightarrow stars$	0.18
$V \rightarrow saw$	1.0	$NP \rightarrow telescopes$	0.1

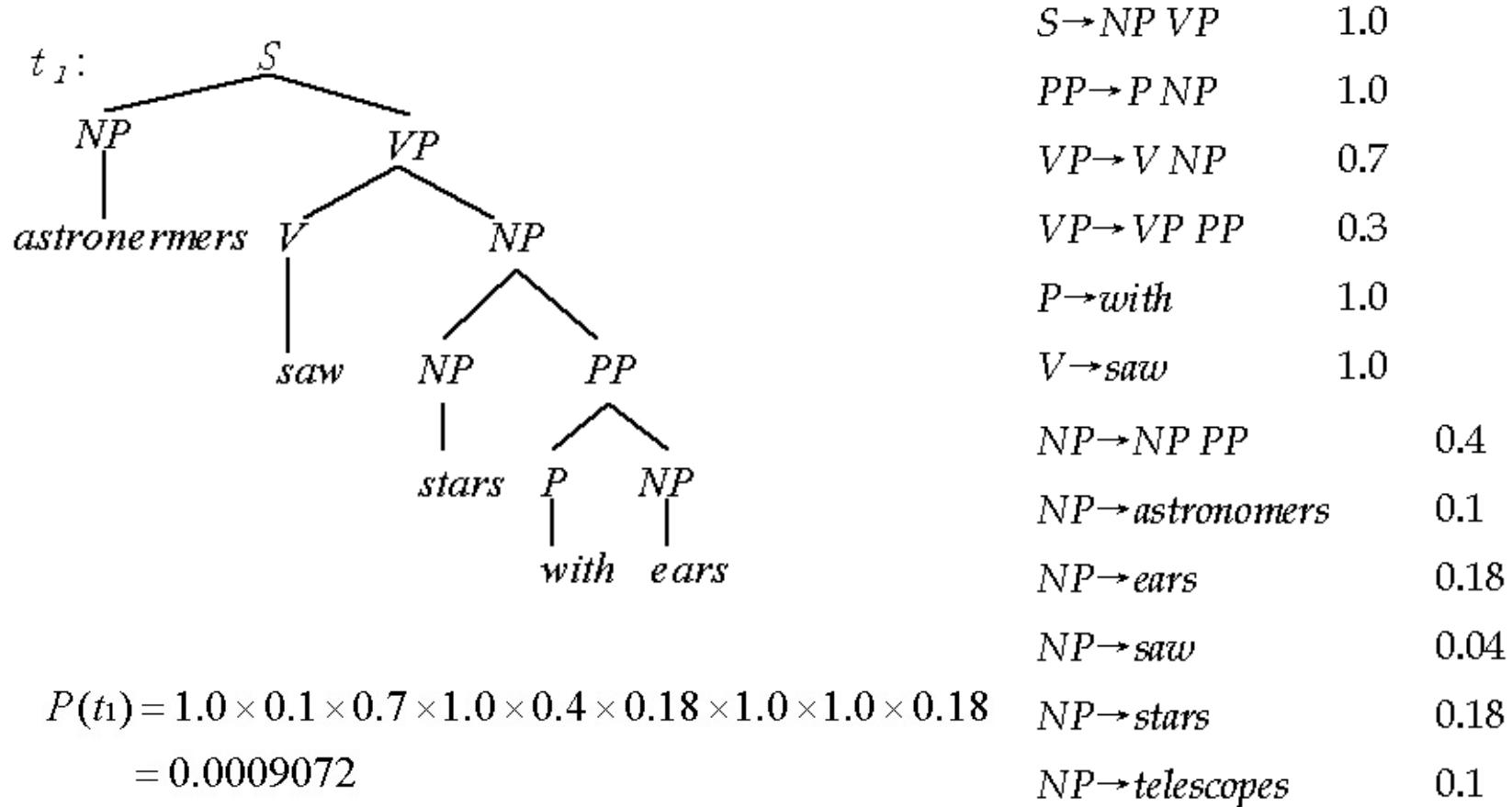
# 利用PCFG计算分析树的概率

$$P(t, S) = \prod_{i=1..n} P(r_i)$$

❖ 句子“astronomers saw stars with ears”的分析树



# 利用PCFG计算分析树的概率



# 利用PCFG计算分析树的概率

$t_2:$		
		$S \rightarrow NP\ VP$ 1.0
		$PP \rightarrow P\ NP$ 1.0
		$VP \rightarrow V\ NP$ 0.7
		$VP \rightarrow VP\ PP$ 0.3
		$P \rightarrow with$ 1.0
		$V \rightarrow saw$ 1.0
		$NP \rightarrow NP\ PP$ 0.4
		$NP \rightarrow astronomers$ 0.1
		$NP \rightarrow ears$ 0.18
		$NP \rightarrow saw$ 0.04
		$NP \rightarrow stars$ 0.18
		$NP \rightarrow telescopes$ 0.1
$P(t_2) =$	$1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18$	
	$= 0.0006804$	

# PCFG 用于句法分析

- 基于PCFG可以计算分析树的概率值。
- 若一个句子有多个分析树，可以依据概率值对所有的分析树进行排序。
- PCFG可以用来进行句法排歧。面对多个分析结果，选择概率最大者为最终分析结果。

# PCFG 用作语言模型

- 基于概率上下文无关文法，一个句子  $w_{1m}$  的概率为：

$$P(S) = \sum_t P(S, t)$$

- 句子 “astronomers saw stars with ears”的概率
  - $P(S) = P(t1) + P(t2) = 0.0009072 + 0.0006804 = 0.0015876$
- PCFG提供了一种统计语言模型，同  $n$ -gram 模型相比，基于 PCFG的语言模型考虑了句子的结构信息，而  $n$ -gram 模型则认为句子是线性结构。

# PCFG的基本问题

- ① 给定一部概率上下文无关文法  $G$ , 如何计算句子  $S$  的概率? 即计算  $P(S|G)$  的问题。 (语言模型)
- ② 给定一部概率上下文无关文法  $G$  以及句子  $S$ , 最为可能的分析树是什么, 即计算  $\arg \max_t P(t|S, G)$  的问题。 (句法分析)
- ③ 如何为文法规则选择概率, 使得训练句子的概率最大? 即计算  $\arg \max_G P(S, G)$  的问题。 (模型训练)
- 可以通过计算每个分析树的概率, 然后以求和或求最大值的方式解决上述第①、②个问题, 缺点是效率不高.

# 向内变量和向外变量

- 为了有效解答PCFG的三个问题，定义向外变量(outside variable)和向内变量(inside variable)。
  - 向外变量

$$\alpha_A(p, q) = P(S \xrightarrow{*} w_1 w_2 \dots w_{p-1} A w_{q+1} \dots w_n) = P(w_1 w_2 \dots w_{p-1} A w_{q+1} \dots w_n | S)$$

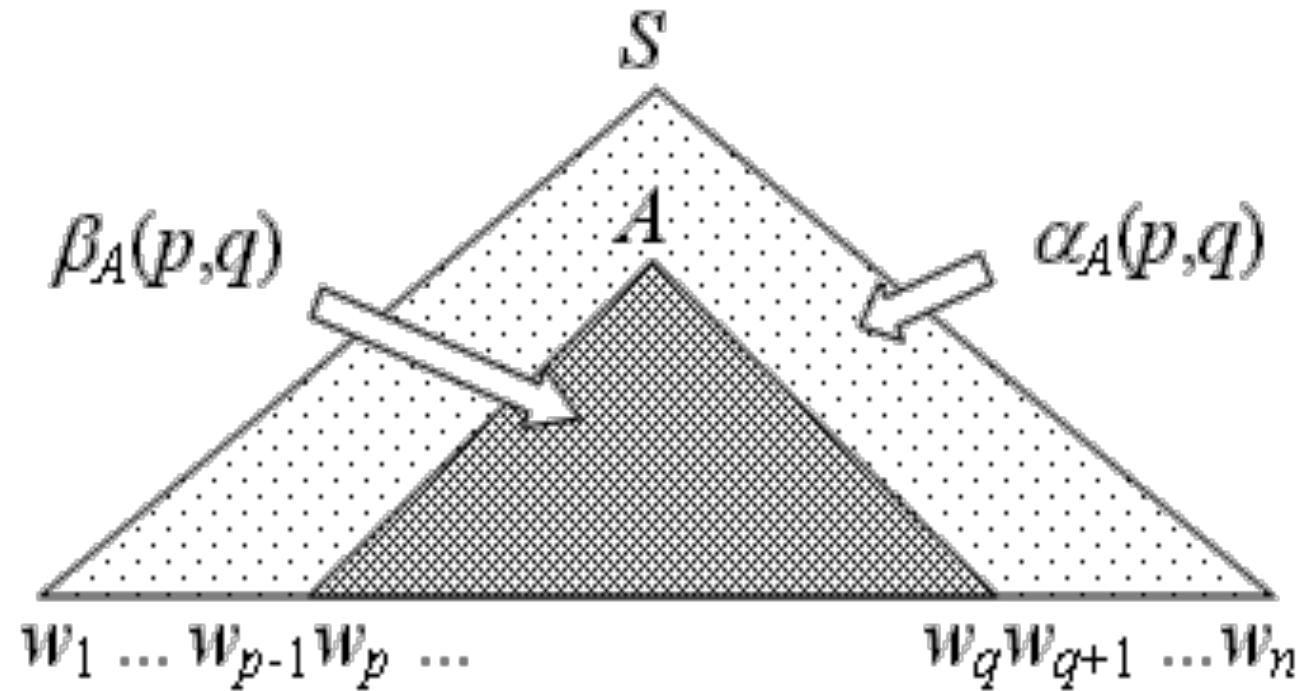
文法开始符号 $S$ 推导出句型 $w_1 w_2 \dots w_{p-1} A w_{q+1} \dots w_n$ 的概率。

- 向内变量

$$\beta_A(p, q) = P(A \xrightarrow{*} w_p w_{p+1} \dots w_q) = P(w_p w_{p+1} \dots w_q | A)$$

非终结符号 $A$ 推导出句子中子串 $w_p w_{p+1} \dots w_q$ 的概率，或者说以 $A$ 为根、叶子为 $w_p w_{p+1} \dots w_q$ 的所有子树的概率。

# 向内变量和向外变量



# 计算句子 $w_{1m}$ 的概率

- 向内变量和句子概率的关系

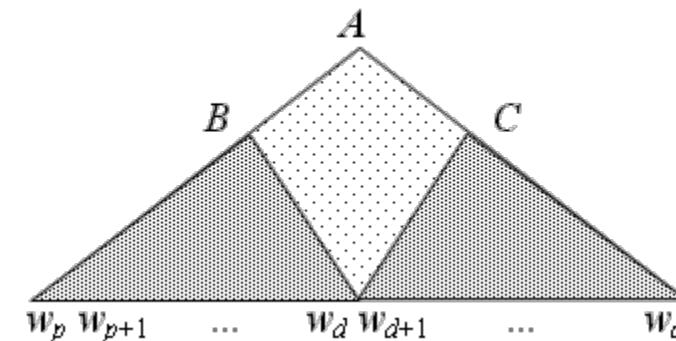
$$\beta_S(1, n) = P(S \xrightarrow{*} w_1 w_2 \dots w_n)$$

- 另有

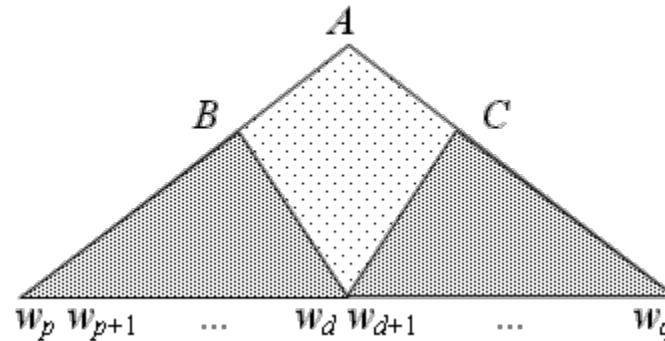
$$\beta_A(k, k) = P(A \xrightarrow{*} w_k)$$

# 向内算法(Inside Algorithm)

- 如何计算  $\beta_A(p, q)$ ，其中  $p < q$
- 因为限制文法为 Chomsky 范式，因此第一条使用的重写规则必为
  - $A \rightarrow BC$
- 子串  $w_{pq}$  一定在某个位置  $d$  被分成两个部分，使得  $B$  支配子串  $w_{pd}$ ，而  $C$  支配子串  $w_{(d+1)q}$ ，



# 向内算法



$$\begin{aligned}
 P(A \Rightarrow BC \xrightarrow{*} w_p w_{p+1} \dots w_d w_{d+1} \dots w_q) &= P(A \Rightarrow BC) P(B \xrightarrow{*} w_p w_{p+1} \dots w_d) P(C \xrightarrow{*} w_{d+1} w_{d+2} \dots w_q) \\
 &= P(A \rightarrow BC) \beta_B(p, d) \beta_C(d+1, q)
 \end{aligned}$$

而计算  $\beta_A(p, q)$  应考虑所有可能的以  $A$  为左部的重写规则，而选定重写规则后，也应考虑将  $w_p w_{p+1} \dots w_q$  分割成两个子串的不同情况，即要考虑为  $d$  做所有可能的选择。故有：

$$\beta_A(p, q) = \sum_{B,C} \sum_d P(A \Rightarrow BC \xrightarrow{*} w_p w_{p+1} \dots w_d w_{d+1} \dots w_q)$$

$$\beta_A(p, q) = \sum_{B,C} \sum_{d=p}^{q-1} P(A \rightarrow BC) \beta_B(p, d) \beta_C(d+1, q)$$

# 向内算法

① 初始化

$$\beta_A(k, k) = P(A \rightarrow w_k)$$

② 归纳计算  $\beta_A(p, q)$  , 其中  $p < q$

$$\beta_A(p, q) = \sum_{B, C} \sum_{d=p}^{q-1} P(A \rightarrow BC) \beta_B(p, d) \beta_C(d+1, q)$$

③ 归纳终止

$$P(w_1 w_2 \dots w_n) = \beta_S(1, n)$$

向内算法自底而上的递归计算句子概率。

# 向内算法计算实例

	1	2	3	4	5
1	$\beta_{NP} = 0.1$		$\beta_S = 0.0126$		$\beta_S = 0.0015876$
2		$\beta_{NP} = 0.04$ $\beta_V = 1.0$	$\beta_{VP} = 0.126$		$\beta_{VP} = 0.015876$
3			$\beta_{NP} = 0.18$		$\beta_{NP} = 0.01296$
4				$\beta_P = 1.0$	$\beta_{PP} = 0.18$
5					$\beta_{NP} = 0.18$
	<i>Astronomers</i>	<i>saw</i>	<i>stars</i>	<i>with</i>	<i>ears</i>

$S \rightarrow NP VP$	1.0	$NP \rightarrow NP PP$	0.4
$PP \rightarrow P NP$	1.0	$NP \rightarrow \text{astronomers}$	0.1
$VP \rightarrow V NP$	0.7	$NP \rightarrow \text{ears}$	0.18
$VP \rightarrow VP PP$	0.3	$NP \rightarrow \text{saw}$	0.04
$P \rightarrow \text{with}$	1.0	$NP \rightarrow \text{stars}$	0.18
$V \rightarrow \text{saw}$	1.0	$NP \rightarrow \text{telescopes}$	0.1

单元格( $p, q$ )内为  
向内概率  $\beta_i(p, q)$

# 基于CKY算法的向内概率计算

```
FUNCTION InsideProbability0
begin
    float inside[1..n][1..|N][1..n] ← 0;
    for k ← 1 to n do
        for each rule  $A \rightarrow w_k \in P$  do
            inside(k, A, k) ←  $P(A \rightarrow w_k)$ ;
    for l ← 2 to n do
        for p ← 1 to n-l+1 do
            for t ← 1 to l-1 do
                q ← p+l-1; d ← p+t-1;
                for each rule  $A \rightarrow BC \in P$  do
                    inside(p, A, q) ← inside(p, A, q) +
                        inside(p, B, d) × inside(d+1, C, q) ×  $P(A \rightarrow BC)$ 
    return inside(1, S, n);
end.
```

# 向外算法(outside algorithm)

- 如何计算  $\alpha_A(p,q)$ ?

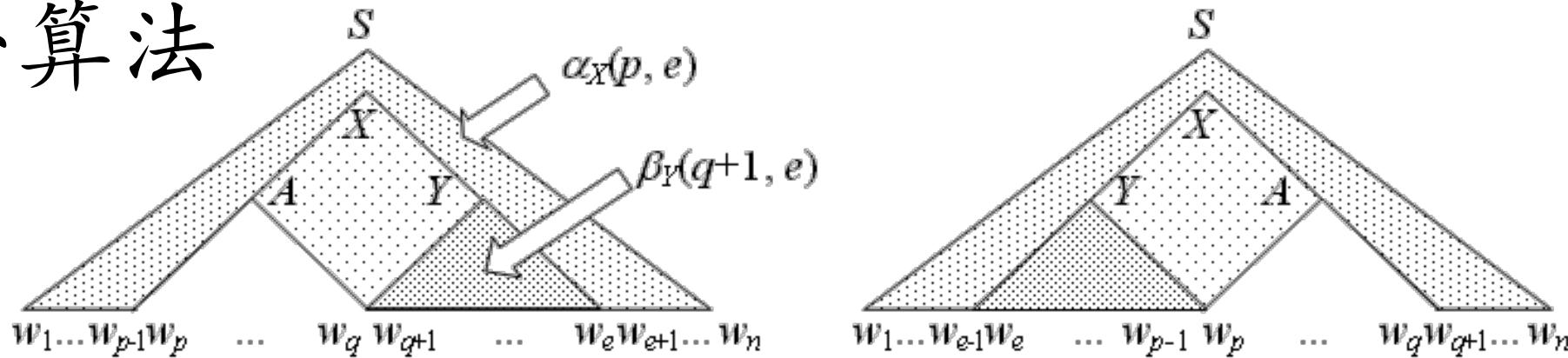
向外变量  $\alpha_A(p,q)$  指的是  $S$  推导出句型  $w_1w_2\dots w_{p-1}Aw_{q+1}\dots w_n$  的概率。因为限制文法是乔姆斯基范式，因此必然存在重写规则  $X \rightarrow AY$  或重写规则  $X \rightarrow YA$ ，使得：

$$S \xrightarrow{*} w_1w_2\dots w_{p-1}Xw_{e+1}\dots w_n \Rightarrow w_1w_2\dots w_{p-1}AYw_{e+1}\dots w_n \text{ 且 } Y \xrightarrow{*} w_{q+1}w_{q+2}\dots w_e$$

或者，

$$S \xrightarrow{*} w_1w_2\dots w_{e-1}Xw_{q+1}\dots w_n \Rightarrow w_1w_2\dots w_{e-1}YAw_{q+1}\dots w_n \text{ 且 } Y \xrightarrow{*} w_e w_{e+1}\dots w_{p-1}$$

# 向外算法



若结点  $A$  作为父结点  $X$  的左子女，即：

$$\begin{aligned} P(S \xrightarrow{*} w_1 \dots w_{p-1} X w_{e+1} \dots w_n &\Rightarrow w_1 \dots w_{p-1} A Y w_{e+1} \dots w_n, Y \xrightarrow{*} w_{q+1} \dots w_e) \\ &= \alpha_X(p, e) P(X \rightarrow AY) \beta_Y(q+1, e) \end{aligned}$$

同理，若结点  $A$  作为父结点  $X$  的右子女，则有：

$$\begin{aligned} P(S \xrightarrow{*} w_1 \dots w_{e-1} X w_{q+1} \dots w_n &\Rightarrow w_1 \dots w_{e-1} YA w_{q+1} \dots w_n, Y \xrightarrow{*} w_e \dots w_{p-1}) \\ &= \alpha_X(e, q) P(X \rightarrow YA) \beta_Y(e, p-1) \end{aligned}$$

所以，如何求  $\alpha_A(p, q)$  ? (5min)

# 向外算法

计算  $\alpha_A(p, q)$  时，对于这两种情况，都需要考虑各种可能的重写规则及选择所有可能的  $e$  值，故：

$$\begin{aligned}\alpha_A(p, q) &= \sum_{X, Y} \sum_{e=q+1}^n \alpha_X(p, e) P(X \rightarrow AY) \beta_Y(q+1, e) \\ &\quad + \sum_{X, Y} \sum_{e=1}^{p-1} \alpha_X(e, q) P(X \rightarrow YA) \beta_Y(e, p-1)\end{aligned}$$

因为句法树的根结点总是文法的开始符号，而不会是其它非终结符号，所以有  $P(S \xrightarrow{*} S) = 1$  及  $P(S \xrightarrow{*} X) = 0$ ，即：

$$\alpha_S(1, n) = 1, \quad \alpha_X(1, n) = 0 (X \neq S)$$

此外，根据向内变量及向外变量的定义，有：

$$\begin{aligned}P(S \xrightarrow{*} w_1 w_2 \dots w_{p-1} A w_{q+1} \dots w_n \xrightarrow{*} w_1 w_2 \dots w_{p-1} w_p \dots w_q w_{q+1} \dots w_n) \\ = P(S \xrightarrow{*} w_1 w_2 \dots w_{p-1} A w_{q+1} \dots w_n) P(A \xrightarrow{*} w_p w_{p+1} \dots w_q) \\ = \alpha_A(p, q) \beta_A(p, q)\end{aligned}$$

# 向外算法

向外算法自顶向下  
递归计算句子的概率

(1) 初始化, 令

$$\alpha_A(1, n) = \begin{cases} 1, & \text{if } A = S \\ 0, & \text{if } A \neq S \end{cases}$$

(2) 归纳计算, 对所有的  $p, q$ , 且  $p < q$ , 令

$$\begin{aligned} \alpha_A(p, q) = & \sum_{X, Y} \sum_{e=q+1}^n \alpha_X(p, e) P(X \rightarrow AY) \beta_Y(q+1, e) \\ & + \sum_{X, Y} \sum_{e=1}^{p-1} \alpha_X(e, q) P(X \rightarrow YA) \beta_Y(e, p-1) \end{aligned}$$

(3) 归纳终止, 对任意非终结符号

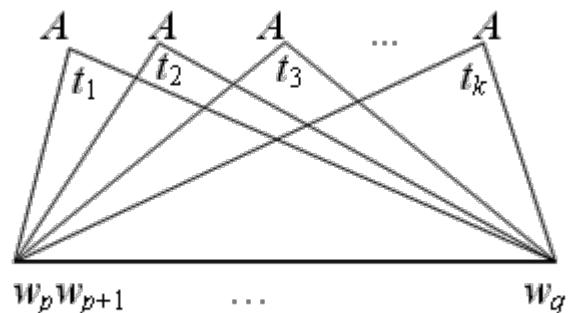
$$P(w_1 w_2 \dots w_n) = \sum_A \alpha_A(p, q) \beta_A(p, q)$$

# 基于CKY算法的向外概率计算

```
FUNCTION OutsideProbability
begin
    float outside[1..n][1..|N|][1..n]:=0;
    outside[1, S, n+1] ← 1;
    for l ← n down to 2 do
        for s ← 1 to n-l+1 do
            for t ← 1 to l-1 do
                p1←s; e1←s+l-1; q1←s+t-1;
                p2←s+t; e2←s; q2←s+l-1;
                for each rule A→BC ∈ P do
                    outside(p1, B, q1) = outside(p1, B, q1) +
                        outside(p1, A, e1) × inside(q1+1, C, e1) × P(A→BC)
                    outside(p2, C, q2) = outside(p2, C, q2) +
                        outside(e2, A, q2) × inside(e2, B, p2-1) × P(A→BC)
    end.
```

# 寻找最佳的分析树

- PCFG的第二个基本问题是在给定文法 $G$ 和句子 $w_{1m}$ 的前提下，如何有效找出最为可能的分析树，这可以通过韦特比算法求得。
- 韦特比变量  $\delta_A(p, q)$
- 以 $A$ 为根并且叶结点是 $w_p w_{p+1} \dots w_q$ 的所有子树中概率最大的子树的概率。



$$\delta_A(p, q) = \max (P(t_1), P(t_2), \dots P(t_k))$$

# 韦特比算法(Viterbi Algorithm)

(1) 初始化, 对所有的  $k$  ( $1 \leq k \leq n$ ), 令

$$\delta_A(k, k) = P(A \rightarrow w_k)$$

(2) 归纳计算, 对所有的  $p, q$ , 且  $p < q$ , 令

$$\delta_A(p, q) = \max_{\substack{p \leq d < q \\ B, C \in N}} P(A \rightarrow BC) \delta_B(p, d) \delta_C(d + 1, q)$$

$$\psi_A(p, q) = \arg \max_{(B, C, d)} P(A \rightarrow BC) \delta_B(p, d) \delta_C(d + 1, q)$$

(3) 归纳终止

$$P(\hat{t}) = \delta_s(1, n)$$

(4) 根据数组  $\psi_A(p, q)$  中记录的信息构造概率最大的分析树  $\hat{t}$ 。

a)  $\hat{t}$  的根结点为  $S$ 。

b) 若  $A$  是  $\hat{t}$  的内部结点(非叶子结点), 并且若  $\psi_A(p, q) = (B, C, d)$ , 则  
 $A$  的左子女是  $B$ , 右子女是  $C$ 。且  $B$  支配子串  $w_p w_{p+1} \dots w_d$ ,  $C$  支配子串  
 $w_{d+1} w_{d+2} \dots w_q$ 。

# 模型训练

- 如何为文法规则选择概率，使得训练句子的概率最大？也就是如何得到文法规则的概率的问题。
- 有指导训练
- 无指导训练，向内向外算法

# 模型训练

- 有指导的训练
- 树库(Treebank)，是标记了句法树结构的语料库。

$$\hat{P}(A \rightarrow BC) = \frac{C(A \rightarrow BC)}{\sum_{\gamma} C(A \rightarrow \gamma)} \quad \hat{P}(A \rightarrow w) = \frac{C(A \rightarrow w)}{\sum_{\gamma} C(A \rightarrow \gamma)}$$

- 树库的构建的工作量巨大，耗时耗力，但在没有可靠的无指导训练技术的前提下，树库的构造必须进行
- 美国宾夕法尼亚大学一直致力于树库的构建工作，其构建的树库被称作 *Penn Treebank*。其中英文树库规模较大、汉语树库的规模较小
- *Penn treebank*尽管规模很小，但为统计句法分析研究 提供了一个很好的基础。

# 向内向外算法(inside-outside算法)

- 无指导训练算法是IO算法
- 同Baum-Welch 算法类似， IO算法也是一个反复迭代、逐步求精的算法。
- 通常要首先给定一组不准确的参数，以反复迭代计算的方式调整模型参数，最终使参数稳定在一个可以接受的精度。
- IO算法不能保证求得最优模型，一般能得到一个局部最优模型。

# 向内向外算法的基本原理

- 没有树结构，无法准确统计：
- $C(A \rightarrow BC)$ 、 $C(A \rightarrow w)$
- 通过CKY算法，可得到给定句子的所有树结构。基于所有树结构，计算规则的期望频次。

$$C'(A) = \sum P(t|S) C(A)$$

$$C'(A \rightarrow BC) = \sum P(t|S) C(A \rightarrow BC)$$

$$C'(A \rightarrow w) = \sum P(t|S) C(A \rightarrow w)$$

- 基于上述期望频次，进行参数估计

$$P'(A \rightarrow BC) = C'(A \rightarrow BC) \div C'(A)$$

$$P'(A \rightarrow w) = C'(A \rightarrow w) \div C'(A)$$

# 向内向外算法的基本原理

- 如何计算  $P(t|S)$ ?
- 前提是参数已知。"蛋生鸡、鸡生蛋"的问题。
- 循环迭代、逐步求精
- 首先给定一组初始参数；
  - 循环，直到得到一组合理的参数 基于当前参数，
  - 计算  $P(t|S)$
  - 计算  $C'(A)$ 、 $C'(A \rightarrow BC)$  以及  $C'(A \rightarrow w)$
  - 计算新参数
- 计算实例
- 效率问题：逐棵计算树的概率

首先，非终结符号  $A$  在句法树中支配子串  $w_p w_{p+1} \dots w_q$  的期望次数，可由下面的公式给出：

$$\begin{aligned}
 & P(A \xrightarrow{*} w_p w_{p+1} \dots w_q | S \xrightarrow{*} w_1 w_2 \dots w_n) \\
 &= \frac{P(A \xrightarrow{*} w_p w_{p+1} \dots w_q, S \xrightarrow{*} w_1 w_2 \dots w_n)}{P(S \xrightarrow{*} w_1 w_2 \dots w_n)} \\
 &= \frac{P(S \xrightarrow{*} w_1 w_2 \dots w_{p-1} A w_{q+1} \dots w_n \xrightarrow{*} w_1 w_2 \dots w_{p-1} w_p \dots w_q w_{q+1} \dots w_n)}{P(S \xrightarrow{*} w_1 w_2 \dots w_n)} \\
 &= \frac{\alpha_A(p, q) \beta_A(p, q)}{P(S \xrightarrow{*} w_1 w_2 \dots w_n)} \quad (\#1\#)
 \end{aligned}$$

令， $\pi = P(S \xrightarrow{*} w_1 w_2 \dots w_n)$ ，则：

$$P(A \xrightarrow{*} w_p w_{p+1} \dots w_q | S \xrightarrow{*} w_1 w_2 \dots w_n) = \frac{\alpha_A(p, q) \beta_A(p, q)}{\pi}$$

考虑到所有可能的  $p < q$ ，则非终结符号  $A$  在句法树中出现的期望次数为：

$$P(A \text{在句法树中出现}) = \sum_{p=1}^n \sum_{q=p}^n \frac{\alpha_A(p, q) \beta_A(p, q)}{\pi}$$

(#2#)

同理，考虑重写规则  $A \rightarrow BC$  在句法树中出现的期望次数：

$$\begin{aligned} & P(A \Rightarrow BC \xrightarrow{*} w_p w_{p+1} \dots w_q | S \xrightarrow{*} w_1 w_2 \dots w_n) \\ &= \frac{P(A \Rightarrow BC \xrightarrow{*} w_p w_{p+1} \dots w_q, B \xrightarrow{*} w_p \dots w_d, C \xrightarrow{*} w_{d+1} \dots w_q, S \xrightarrow{*} w_1 w_2 \dots w_n)}{P(S \xrightarrow{*} w_1 w_2 \dots w_n)} \\ &= \frac{\sum_{d=p}^{q-1} \alpha_A(p, q) P(A \rightarrow BC) \beta_B(p, d) \beta_C(d+1, q)}{\pi} \end{aligned}$$

考虑到所有可能的  $p < q$ ，重写规则  $A \rightarrow BC$  在句法树中出现的期望次数为

$$P(A \rightarrow BC \text{在句法树中出现}) = \frac{\sum_{p=1}^{n-1} \sum_{q=p+1}^n \sum_{d=p}^{q-1} \alpha_A(p, q) P(A \rightarrow BC) \beta_B(p, d) \beta_C(d+1, q)}{\pi}$$

因此有：

$$\begin{aligned} \hat{P}(A \rightarrow BC) &= \frac{P(A \rightarrow BC \text{在句法树中出现})}{P(A \text{在句法树中出现})} \\ &= \frac{\sum_{p=1}^{n-1} \sum_{q=p+1}^n \sum_{d=p}^{q-1} \alpha_A(p, q) P(A \rightarrow BC) \beta_B(p, d) \beta_C(d+1, q)}{\sum_{p=1}^m \sum_{q=p}^m \alpha_A(p, q) \beta_A(p, q)} \quad (\#4#) \end{aligned}$$

# 向内向外算法的基本原理

对  $A \rightarrow w$ , 也可做类似的推导, 若  $w_k = w$ , 有:

$$\begin{aligned}
 & P(A \rightarrow w_k | S \Rightarrow w_1 w_2 \dots w_n) \\
 &= \frac{P(A \rightarrow w_k, S \Rightarrow w_1 w_2 \dots w_{k-1} A w_{k+1} \dots w_n)}{P(S \Rightarrow w_1 w_2 \dots w_n)} \\
 &= \frac{\alpha_A(k, k) \beta_A(k, k)}{\pi}
 \end{aligned}$$

考虑到所有可能的位置  $k$ , 根据  $w_k$  是否为  $w$ , 则:

$$P(A \rightarrow w \text{ 在句法树中出现}) = \frac{\sum_{k=1}^n \alpha_A(k, k) \beta_A(k, k) \delta(w_k, w)}{\pi}$$

$$\begin{aligned}
 \hat{P}(A \rightarrow w) &= \frac{P(A \rightarrow w \text{ 在句法树中出现})}{P(A \text{ 在句法树中出现})} \\
 &= \frac{\sum_{k=1}^n \alpha_A(k, k) \beta_A(k, k) \delta(w_k, w)}{\sum_{p=1}^m \sum_{q=p}^m \alpha_A(p, q) \beta_A(p, q)}
 \end{aligned}$$

(#5#)

# IO算法描述

- EM算法特例
- 不保证收敛到全局最优点
- 对初始参数较敏感
  - (1) 任意设定一组概率文法参数  $\hat{P}_0(A \rightarrow BC)$  及  $\hat{P}_0(A \rightarrow w)$
  - (2) 令  $i \leftarrow 1$
  - (3) 循环执行下面的步骤，直到文法参数收敛
    - 1) E-Step: 基于  $\hat{P}_{i-1}(A \rightarrow BC)$  及  $\hat{P}_{i-1}(A \rightarrow w)$  计算公式(#1#)、(#2#)、(#3#)的值
    - 2) M-Step: 将 E-Step 的计算结果代入公式(#4#)和(#5#)，得到一组更新的参数  $\hat{P}_i(A \rightarrow BC)$  及  $\hat{P}_i(A \rightarrow w)$
    - 3) 令  $i \leftarrow i+1$
  - (4) 算法结束，输出概率文法参数

# 基于PCFG的句法分析

- PCFG把概率引入上下文无关文法，将统计方法和规则方法进行了有效的融合，具有十分重要的意义，但是PCFG缺陷也是十分明显的。
- 作为一种统计句法分析方法，基于PCFG的句法分析效果有限。
  - PCFG没有考虑结构之间的依存关系。
  - PCFG没有考虑词汇对句法结构的影响。
  - 需要针对PCFG的句法分析表现出来的缺陷进行改进。

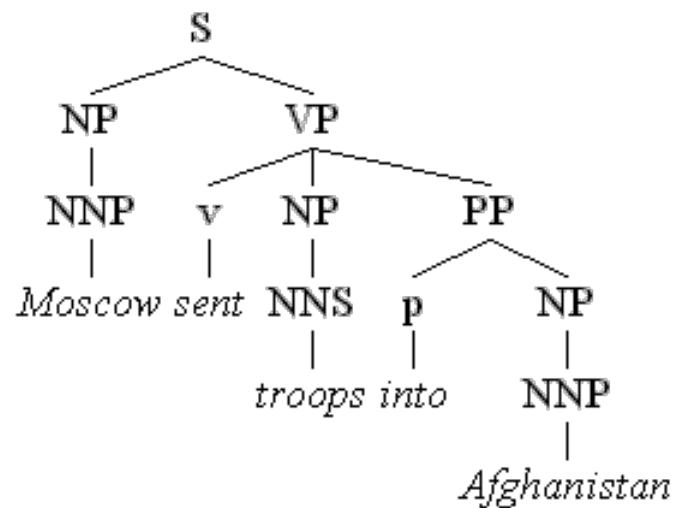
# 结构依存关系

- 代词作为主语的可能性高于作为宾语的可能性
  - 主题与述题
  - 主题通常是已知信息，在句子中做主语
  - 代词用来指代已经陈述过的已知信息
  - 述题引入新信息
  - 统计数据，来自Switchboard corpus:
    - ◆ 陈述句中91%的主语是代词
    - ◆ 陈述句中66% 的直接宾语不是代词

$NP \rightarrow Pron$      $NP \rightarrow Det\ Noun$  的概率应和其在句中所处的位置有关，处在VP前后概率应该不同。

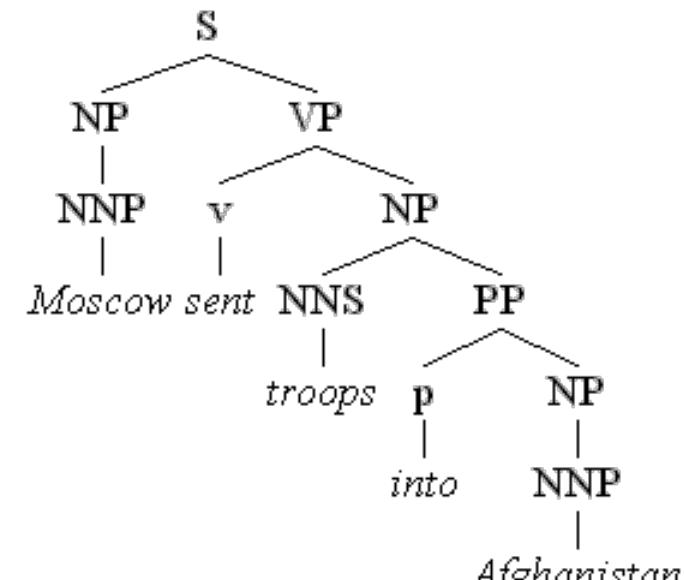
# 词汇依存关系

*Moscow sent troops into Afghanistan*



正确的句法树

*PP可以修饰 VP , PP也  
可以修饰 NP, 但不同的  
动词 , PP修饰NP和VP的  
可能性并不相同。*



错误的句法树

# 基于PCFG的统计语言模型

- 作为一种统计语言模型，其效果甚至还不如n-gram模型以及HMM模型(原因同上)
- the green banana the green time

# 句法分析的评价

## ◆ 标记准确率

$$\text{Labeled Precision} = \frac{\text{number of correct constituents in proposed parse}}{\text{number of constituents in proposed parse}}$$

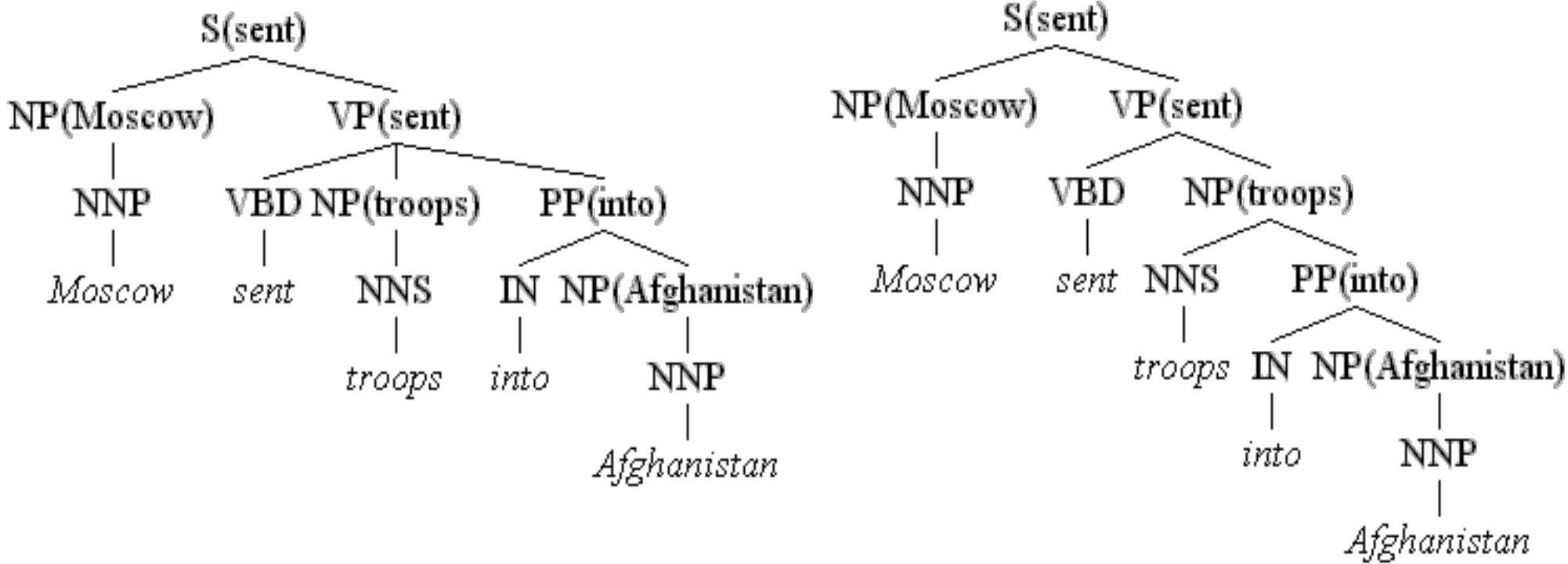
## ◆ 标记召回率

$$\text{Labeled Recall} = \frac{\text{number of correct constituents in proposed parse}}{\text{number of constituents in treebank parse}}$$

## ◆ 括号交叉数

$$\text{Crossing Brackets} = \text{number of constituents which violate constituent boundaries with a constituent in the treebank parse}$$

# 词汇化的句法分析简介



# 词汇化PCFG

$S(\text{sent}) \rightarrow NP(\text{Moscow}) VP(\text{sent})$

$NP(\text{Moscow}) \rightarrow NNP(\text{Moscow})$

$VP(\text{sent}) \rightarrow VBD(\text{sent}) NP(\text{troops}) PP(\text{into})$

$NP(\text{troops}) \rightarrow NNS(\text{troops}) PP(\text{into})$

$PP(\text{into}) \rightarrow IN(\text{into}) NP(\text{Afghansitan})$

$NP(\text{Afghanistan}) \rightarrow NNP(\text{Afghanistan})$

$NNP(\text{Moscow}) \rightarrow \text{Moscow}$

$NNP(\text{Afghanistan}) \rightarrow \text{Afghanistan}$

$IN(\text{into}) \rightarrow \text{into}$

$VBD(\text{sent}) \rightarrow \text{sent}$

$NNS(\text{troops}) \rightarrow \text{troops}$

非终结符号的数量激烈膨胀。

参数估计时，数据稀疏问题十分严重！

# 词汇化PCFG

- 为避免数据稀疏问题，概率计算需要分解
- 在词汇化PCFG中

$$P(h) \rightarrow L_n(l_n) \dots L_1(l_1) H(h) R_1(r_1) \dots R_m(r_m)$$

其中， $H$ 是短语的中心成分

$h$ 是成分的中心词

$L_i$ 和 $R_i$ 分别是中心成分左右的修饰性成分

$l_i$ 和 $r_i$ 分别是左右修饰成分的中心词

- 在左右两端增加 $STOP$
- 令  $L_{n+1} = STOP$     $R_{n+1} = STOP$

# 词汇化PCFG

$$P(L_{n+1}(l_{n+1}) \dots L_1(l_1) H(h) R_1(r_1) \dots R_{m+1}(r_{m+1}) | P(h)) =$$

$$P_h(H | P(h)) \times$$

$$\prod_{i=1 \dots n+1} P_l(L_i(l_i) | L_1(l_1) \dots L_{i-1}(l_{i-1}), P(h), H) \times$$

$$\prod_{j=1 \dots n+1} P_r(R_j(r_j) | L_1(l_1) \dots L_{n+1}(l_{n+1}), R_1(r_1) \dots R_{j-1}(r_{j-1}), P(h), H)$$

- 作如下的独立性假设

$$P_l(L_i(l_i) | L_1(l_1) \dots L_{i-1}(l_{i-1}), P(h), H) = P_l(L_i(l_i) | P(h), H)$$

$$P_r(R_j(r_j) | L_1(l_1) \dots L_{n+1}(l_{n+1}), R_1(r_1) \dots R_{j-1}(r_{j-1}), P(h), H) = P_r(R_j(r_j) | P(h), H)$$

# 词汇化PCFG

$$P(L_{n+1}(l_{n+1}) \dots L_1(l_1) H(h) R_1(r_1) \dots R_{m+1}(r_{m+1}) | P(h)) = P(H|P(h)) \times \\ \prod_{i=1 \dots n+1} P_l(L_i(l_i) | P(h), H) \times \prod_{j=1 \dots m+1} P_r(R_j(r_j) | P(h), H)$$

- 句法树可以视为自顶向下按照三个步骤产生：

- 以概率  $P_H(H | P, h)$  生成短语的中心成分标签；
- 以概率  $\prod_{i=1 \dots n+1} P_l(L_i(l_i) | P, h, H)$  生成中心词左面的修饰性成分，其中

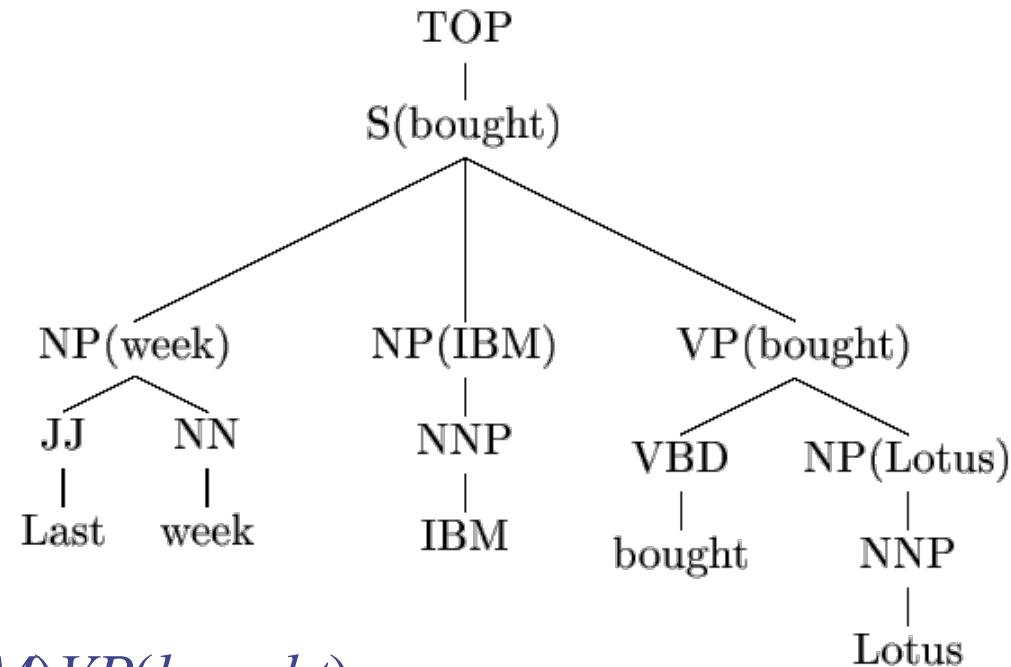
$L_{n+1}(l_{n+1}) = STOP$ 。  $STOP$  可视作一个特殊的非终结符号，模型在生成  $STOP$  标记时结束生成。

- 以概率  $\prod_{i=1 \dots m+1} P_r(R_i(r_i) | P, h, H)$  生成中心词右面的修饰性成分，其中

$R_{m+1}(r_{m+1}) = STOP$ 。

# 词汇化PCFG

- 例子



$S(bought) \rightarrow NP(week)NP(IBM)VP(bought)$

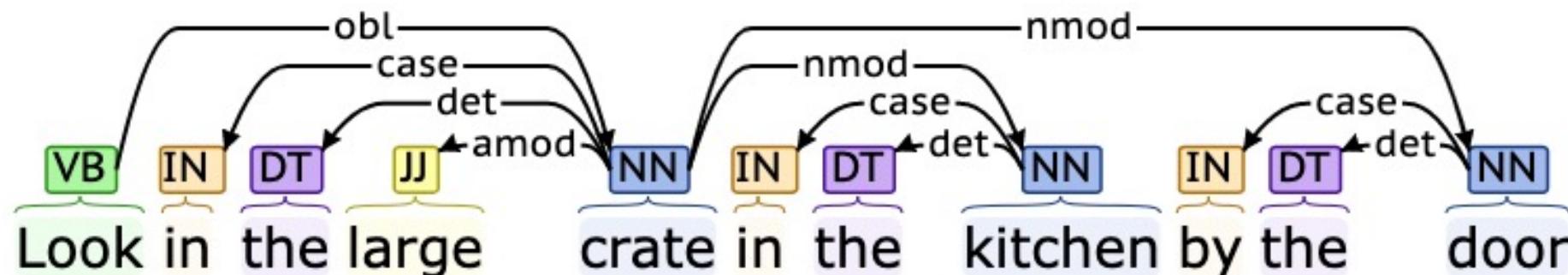
$$\begin{aligned} P_h(VP | S, bought) &\times P_l(NP(IBM) | S, VP, bought) \times \\ P_l(NP(week) | S, VP, bought) &\times P_l(STOP | S, VP, bought) \times \\ P_r(STOP | S, VP, bought) \end{aligned}$$

# 句法分析都有哪些

- 成分分析 (Constituency Parsing)
  - CFG 分析法
  - 基于统计的分析法
- 依存分析 (Dependency Parsing)
  - 规约法 (Transition-based parsing)
  - 图模型法 (Graph-based parsing)

# Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify, attach to, or are arguments of) which other words.



# Why do we need sentence structure?

- Humans communicate complex ideas by composing words together into bigger units to convey complex meanings
- Listeners need to work out what modifies [attaches to] what
- A model needs to understand sentence structure in order to be able to interpret language correctly

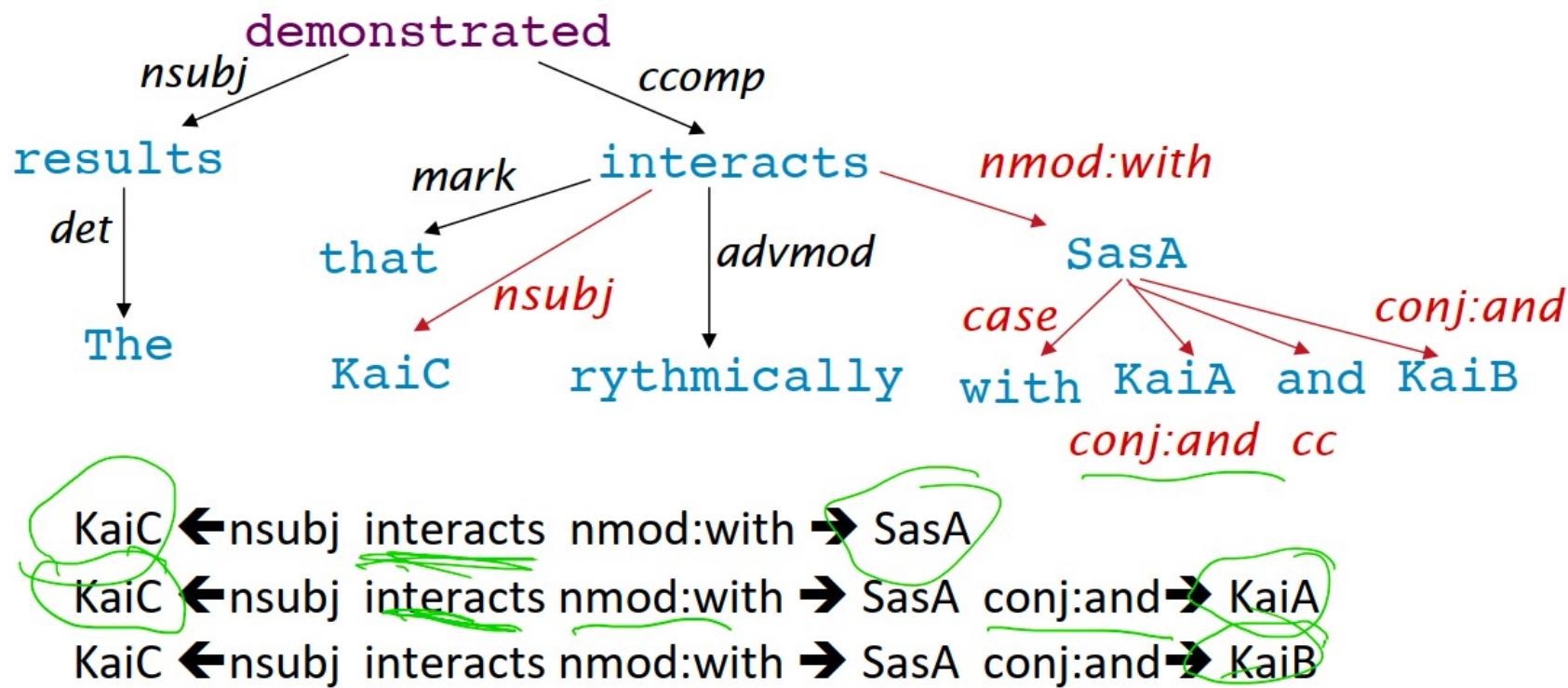
# PP attachment ambiguities multiply

- A key parsing decision is how we ‘attach’ various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations,
- The board approved [its acquisition] [by Royal Trustco Ltd.]  
[of Toronto]  
[for \$27 an share]  
[at its monthly meeting].

# Coordination scope ambiguity

- [Shuttle veteran] and [longtime NASA executive Fred Gregory] appointed to board 2 people
- [Shuttle veteran and longtime NASA executive Fred Gregory] appointed to board 1 people

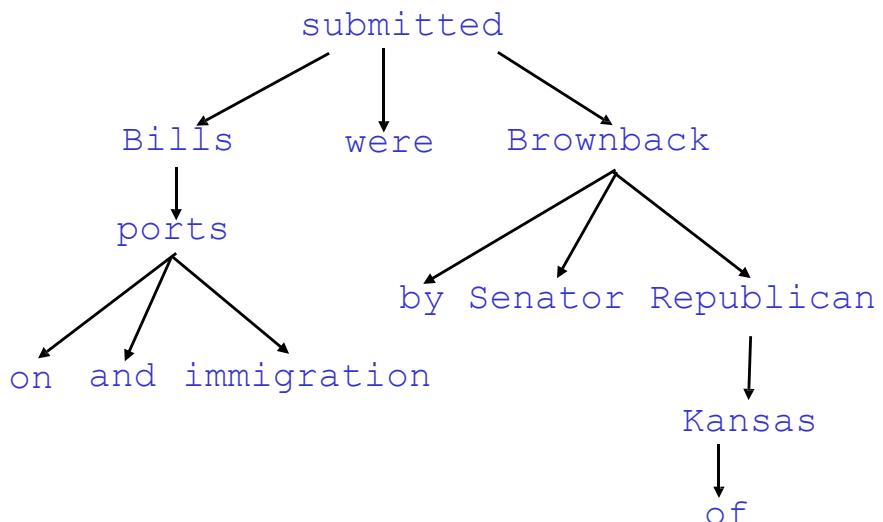
## Dependency paths help extract semantic interpretation – simple practical example: extracting protein-protein interaction



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

## 2. Dependency Grammar and Dependency Structure

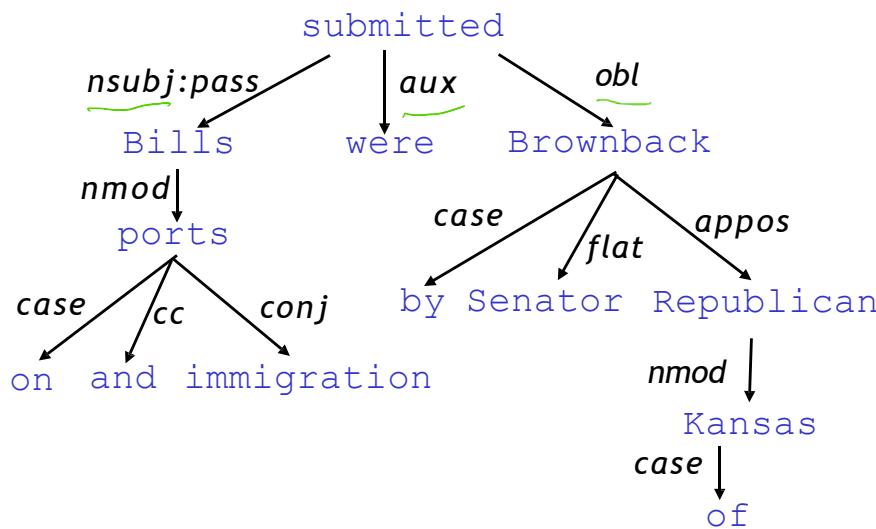
- Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called **dependencies**



# Dependency Grammar and Dependency Structure

- Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)

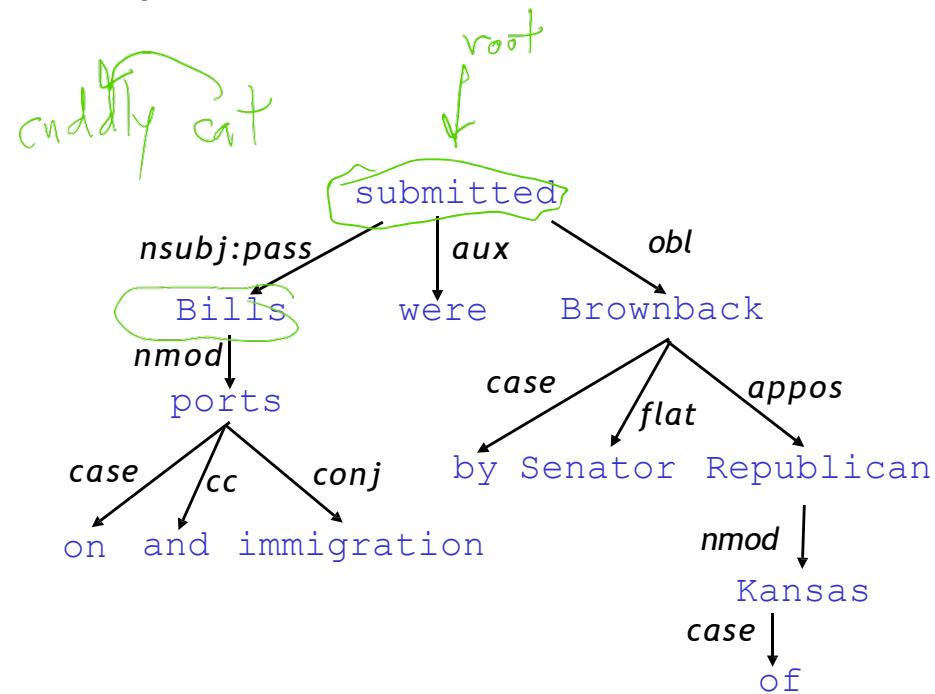


# Dependency Grammar and Dependency Structure

- Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called dependencies

An arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (a connected, acyclic, single-root graph)



# Pāṇini's grammar (c. 5th century BCE)



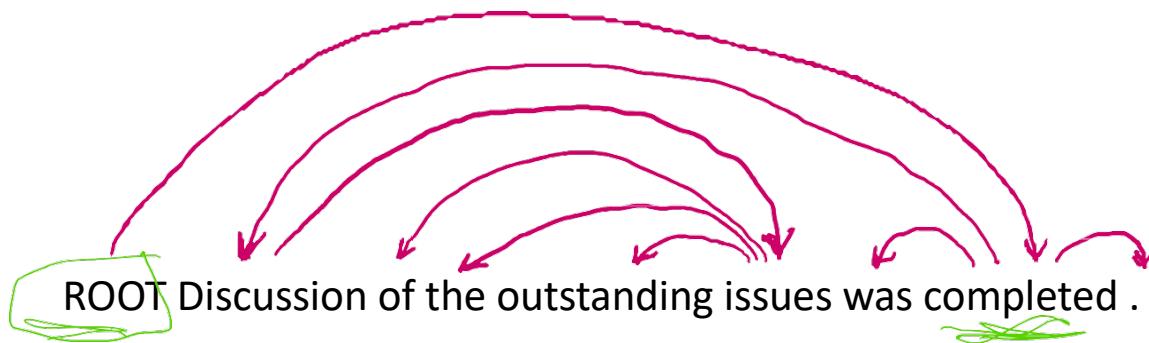
Gallery: <http://wellcomeimages.org/indexplus/image/L0032691.html>

CC BY 4.0 File:Birch bark MS from Kashmir of the Rupavatra Wellcome L0032691.jpg

# Dependency Grammar/Parsing History

- 依存结构的概念可以追溯到很久以前
  - 源自波尼尼的语法(公元前5世纪左右)
  - 第一个千年的阿拉伯语法学家的基本方法
- constituency/CFG是一个新发明
  - 20世纪的发明(R.S. Wells, 1947; 然后是乔姆斯基1953年等)
- 现代依存句法研究通常追溯到吕西安·泰尼尔(1959)
  - 在20世纪的"东方"是主导方法(俄罗斯、中国等)
  - 适用于语序较自由、屈折语言如俄语(或拉丁语!)
- 即使在美国,也被用于最早的一些自然语言处理解析器中:
  - 美国计算语言学的创始人之一大卫·海斯构建了早期(最早?)的依存句法分析器(Hays 1962),并在《语言》杂志上发表了关于依存语法的文章

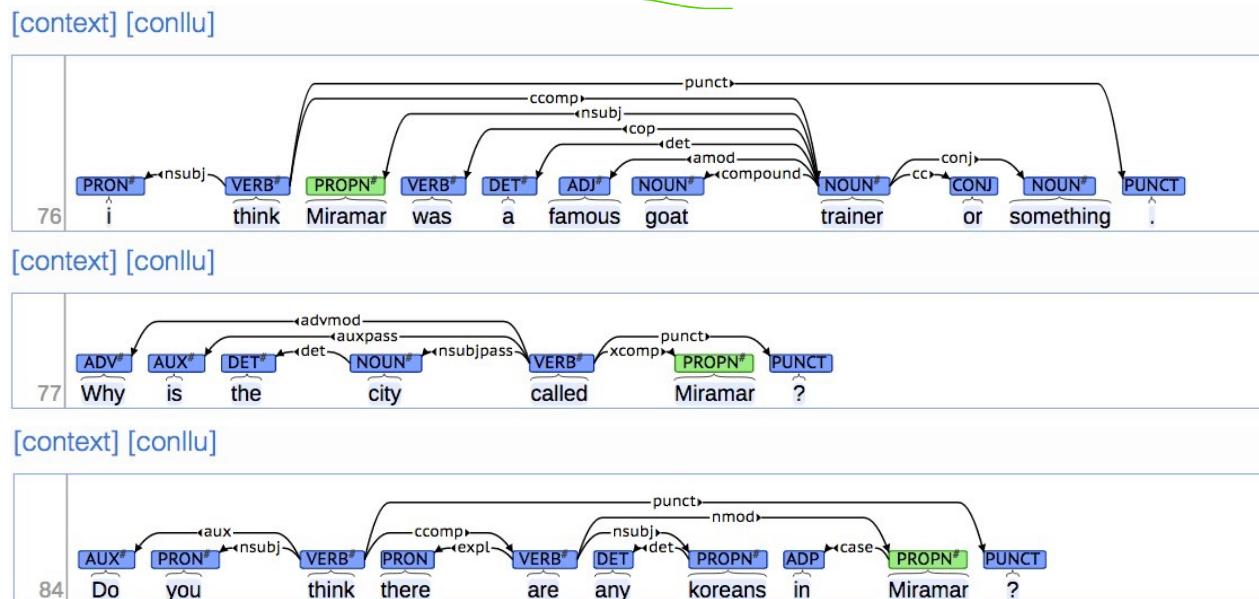
# Dependency Grammar and Dependency Structure



- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent – we follow that convention
- We usually add a fake ROOT so every word is a dependent of precisely 1 other node

# The rise of annotated data & Universal Dependencies treebanks

- Brown corpus (1967; PoS tagged 1979); Lancaster-IBM Treebank (starting late 1980s); Marcus et al. 1993, *The Penn Treebank, Computational Linguistics*;
- Universal Dependencies: <http://universaldependencies.org/>



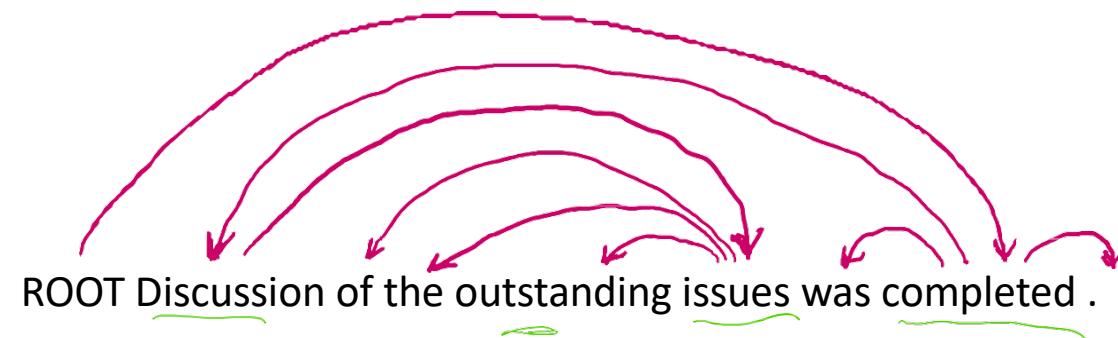
# The rise of annotated data

- 起初,构建树库似乎比(手工)编写语法要慢得多,也没那么有用
- 但是树库给我们带来很多好处
  - 劳动成果的可重用性
    - 可以在其基础上构建许多句法分析器、词性标注器等
    - 语言学研究的宝贵资源
  - 广泛覆盖,而不仅仅是少数直观的语言现象
  - 频率和分布信息
  - 评估自然语言处理系统的一种方式

# Dependency Conditioning Preferences

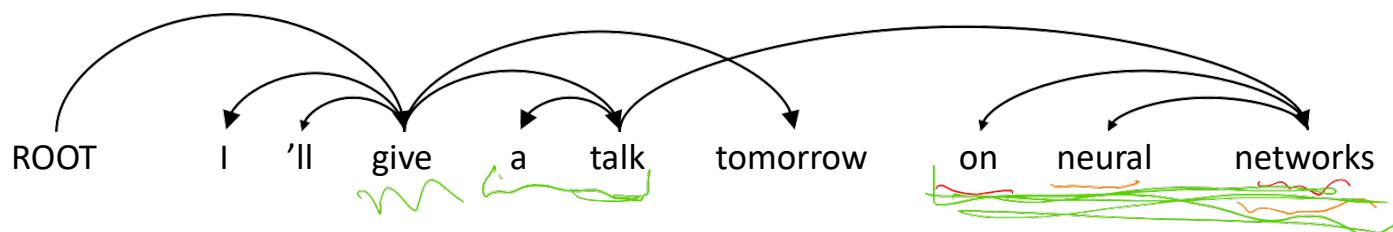
What are the sources of information for dependency parsing?

1. Bilexical affinities (双词亲和力) 依存关系 [discussion → issues] 是合理的
2. Dependency distance (依存距离) 大多数依存关系都存在于相邻的词语之间
3. Intervening material (干扰成分) 依存关系很少跨越中间的动词或标点符号
4. Valency of heads (中心词的配价) 一个中心词通常在其左右两侧分别有多少个依存成分?



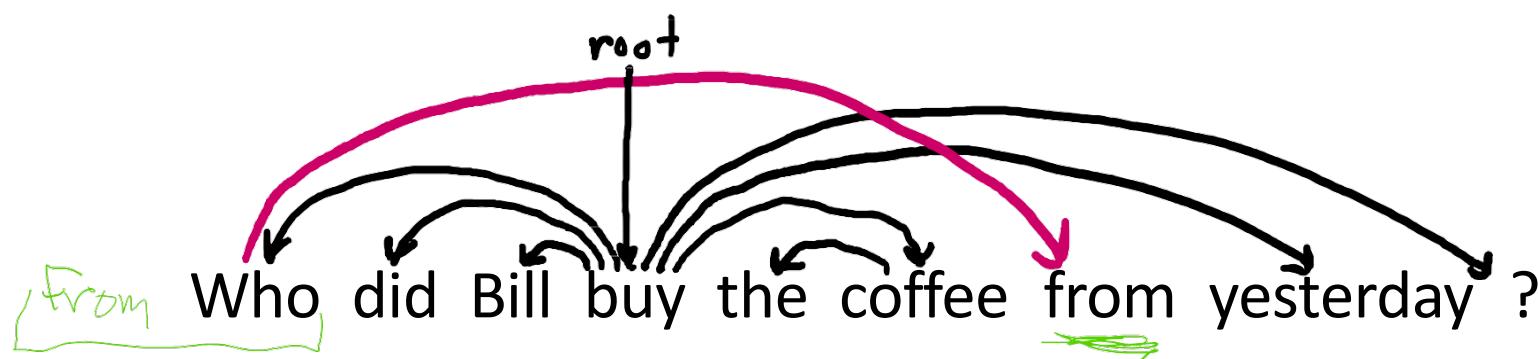
# Dependency Parsing

- 通过为每个词选择它依存于哪个其他词(包括ROOT)，来对一个句子进行句法分析
- 通常有一些约束：
  - 只有一个词是ROOT的依存成分
  - 不希望出现循环依存  $A \rightarrow B, B \rightarrow A$
- 这使得依存关系形成一棵树
- 最后一个问题是否可以交叉(非投射)



# Projectivity

- 投射句法分析的定义：当词语按其线性顺序排列，所有依存弧都位于词语上方时，不存在交叉的依存弧
- 对应于CFG(上下文无关文法)树的依存关系必须是投射的
  - 即，通过将每个范畴的一个子节点作为中心词来形成依存关系
- 大多数句法结构都是如此投射的，但是依存理论通常允许非投射结构来解释移位的成分
  - 如果没有这些非投射依存关系，就不容易正确得到某些结构的语义



# 3. Methods of Dependency Parsing

- 动态规划
  - Eisner (1996) 提出了一个巧妙的算法,其复杂度为 $O(n^3)$ ,通过在两端而不是中间生成带有中心词的句法分析项目
- 图算法
  - 为一个句子创建最小生成树
  - McDonald et al. (2005)的MSTParser使用ML分类器(他使用MIRA进行在线学习,但也可以是其他方法)对依存关系进行独立评分
  - 神经图算法句法分析器: Dozat和Manning (2017)及后续工作——非常成功!
- 约束满足
  - 不满足硬约束的边会被消除。Karlsson (1990)等。
- "基于转移的句法分析"或"确定性依存句法分析"
  - 由良好的机器学习分类器指导依附的贪心选择
  - 例如, MaltParser(Nivre et al. 2008)。事实证明非常有效。

# Greedy transition-based parsing [Nivre 2003]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom-up actions
  - Roughly like “shift” or “reduce” in a shift-reduce parser, but the “reduce” actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack  $\sigma$ , written with top to the right
    - which starts with the ROOT symbol
  - a buffer  $\beta$ , written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs  $A$ 
    - which starts off empty
  - a set of actions



# Basic transition-based dependency parser

**Start:**  $\sigma = [\text{ROOT}]$ ,  $\beta = w_1, \dots, w_n$ ,  $A = \emptyset$

1. Shift             $\sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A$

2. Left-Arc<sub>r</sub>     $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_j, \beta, A \cup \{r(w_j, w_i)\}$

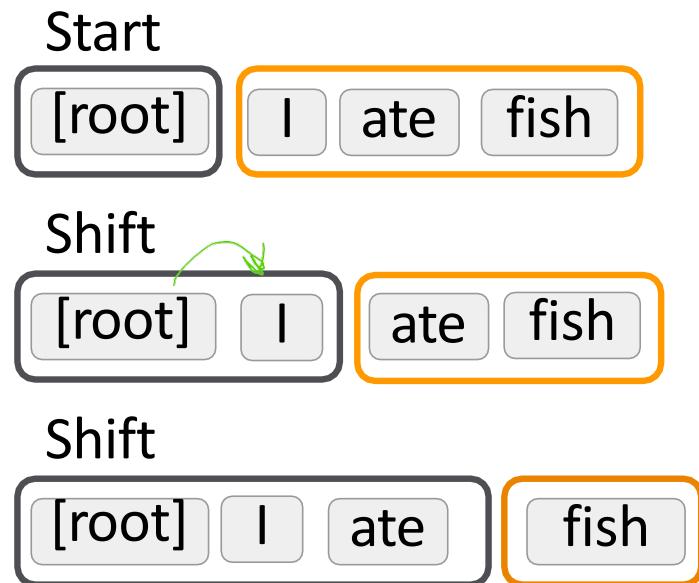
3. Right-Arc<sub>r</sub>     $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_i, \beta, A \cup \{r(w_i, w_j)\}$

**Finish:**  $\sigma = [w]$ ,  $\beta = \emptyset$

# Arc-standard transition-based parser

(there are other transition schemes ...)

- Analysis of “I ate fish”



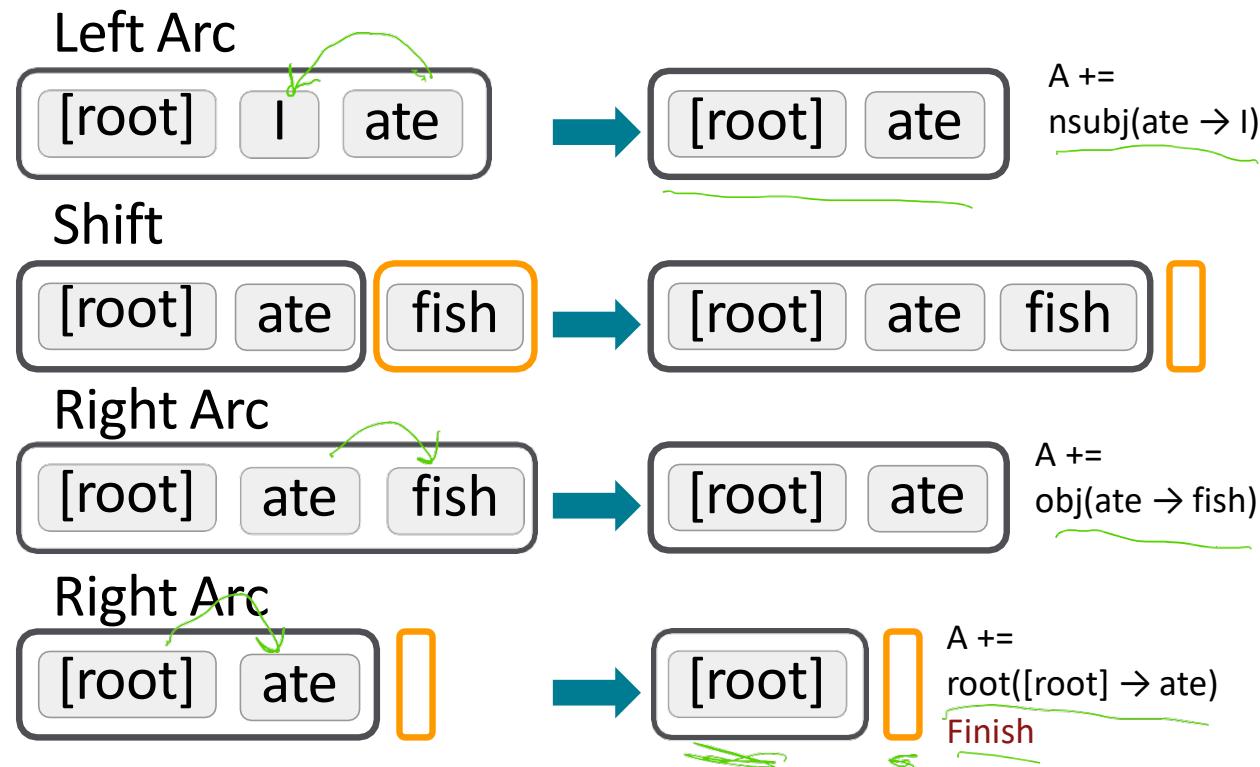
Start:  $\sigma = [\text{ROOT}], \beta = w_1, \dots, w_n, A = \emptyset$

1. Shift  $\sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A$
2. Left-Arc<sub>r</sub>  $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_j, \beta, AU\{r(w_j, w_i)\}$
3. Right-Arc<sub>r</sub>  $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_i, \beta, AU\{r(w_i, w_j)\}$

Finish:  $\sigma = [w], \beta = \emptyset$

# Arc-standard transition-based parser

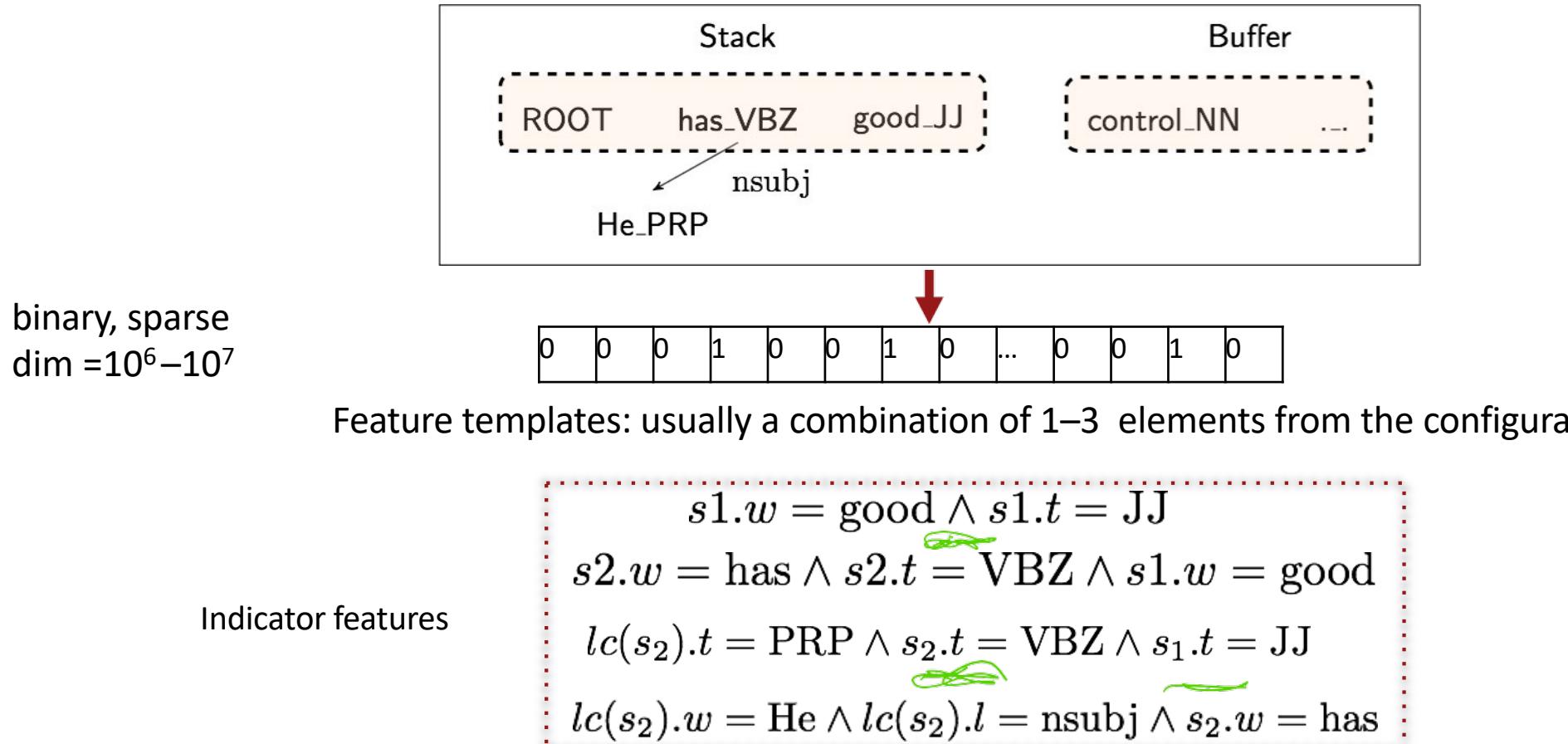
- Analysis of “I ate fish”



# MaltParser [Nivre and Hall 2005]

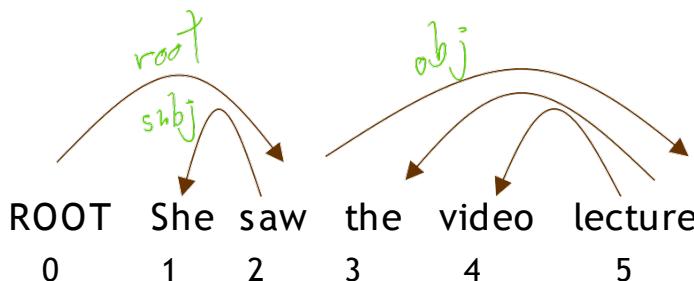
- We have left to explain how we choose the next action 🤖
  - Answer: Stand back, I know machine learning!
- Each action is predicted by a discriminative classifier (e.g., softmax classifier) over each legal move
  - Max of 3 untyped choices; max of  $|R| \times 2 + 1$  when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest form)
  - But you can profitably do a beam search if you wish (slower but better): You keep  $k$  good parse prefixes at each time step
- The model's accuracy is *fractionally* below the state of the art in dependency parsing, but
- It provides **very fast linear time parsing**, with high accuracy – great for parsing the web

# Conventional Feature Representation



# Evaluation of Dependency Parsing: (labeled) dependency accuracy

- labeled attachment score (LAS)
- unlabeled attachment score (UAS)



$$\text{Acc} = \frac{\# \text{ correct deps}}{\# \text{ of deps}}$$
$$\text{UAS} = 4 / 5 = 80\%$$
$$\text{LAS} = 2 / 5 = 40\%$$

Gold			
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

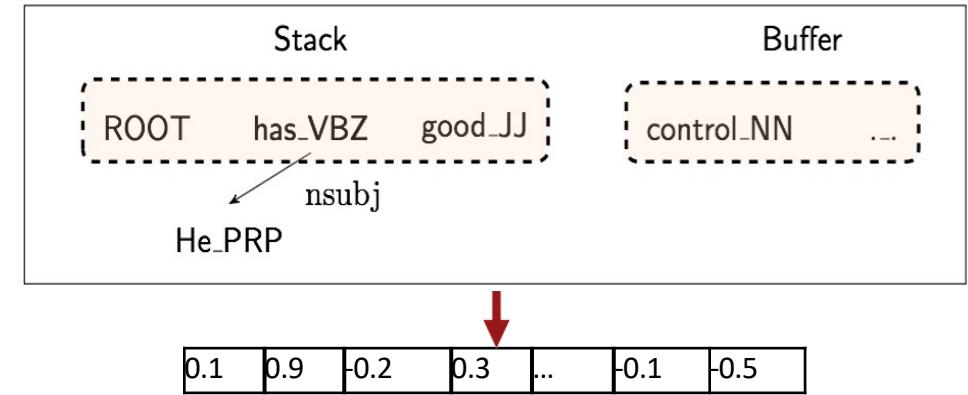
Parsed			
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

# Handling non-projectivity

- 我们介绍的基于弧标准的算法只能构建投射依存树
- 指向中心词的可能方向:
  - 对于非投射弧，直接宣告失败
  - 使用只有投射表示的依存形式
    - CFG只允许投射结构；可以提升违反投射性的中心词
    - 使用投射依存分析算法的后处理器来识别和解决非投射链接
    - 添加额外的转换，以对大多数非投射结构进行建模(例如，添加一个额外的SWAP转换，参考冒泡排序)
    - 换成一种不使用或不需要任何投射性约束的分析机制(例如，基于图的MSTParser或Dozat和Manning(2017))

## 4. Why do we gain from a neural dependency parser? Indicator Features Revisited

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: expensive computation
- Dense
  - $\text{dim} = \sim 1000$  More than 95% of parsing time is consumed by feature computation
- Neural Approach:
  - learn a dense and compact feature representation



$s1.w = \text{good} \wedge s1.t = \text{JJ}$   
 $s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$   
 $lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$   
 $lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$

# A neural dependency parser [Chen and Manning 2014]

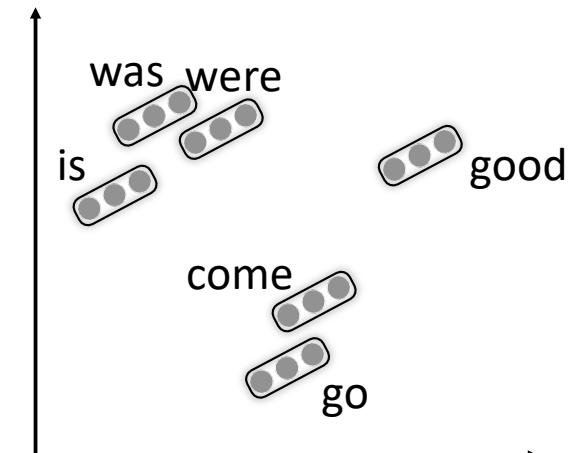
- Results on English parsing to Stanford Dependencies:
  - Unlabeled attachment score (UAS) = head
  - Labeled attachment score (LAS) = head and label

Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	<b>92.3</b>	89.6	8
C & M 2014	92.0	<b>89.7</b>	<b>654</b>

# First win: Distributed Representations

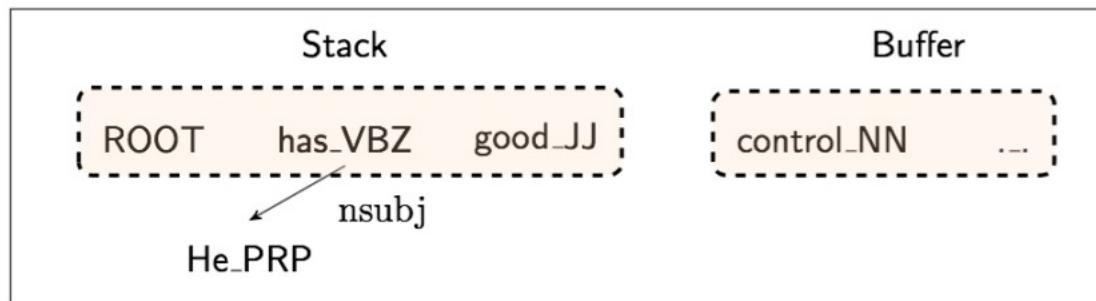
- We represent each word as a  $d$ -dimensional dense vector (i.e., word embedding)
  - Similar words are expected to have close vectors.
- Meanwhile, **part-of-speech tags** (POS) and **dependency labels** are also represented as  $d$ -dimensional vectors.
  - The smaller discrete sets also exhibit many semantical similarities.

NNS (plural noun) should be close to NN (singular noun).  
nummod (numerical modifier) should be close to amod (adjective modifier).



# Extracting Tokens & vector representations from configuration

- We extract a set of tokens based on the stack / buffer positions:



	word	POS	dep.	
s <sub>1</sub>	good	JJ	Ø	
s <sub>2</sub>	has	VBZ	Ø	
b <sub>1</sub>	control	NN	Ø	
lc(s <sub>1</sub> )	Ø	Ø	Ø	
rc(s <sub>1</sub> )	Ø	Ø	Ø	
lc(s <sub>2</sub> )	He	PRP	nsubj	
rc(s <sub>2</sub> )	Ø	Ø	Ø	

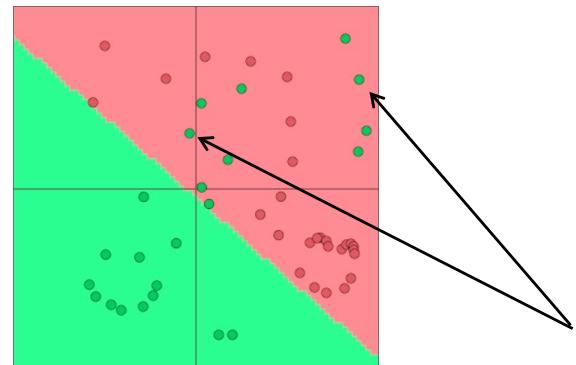
A concatenation of the vector representation of all these is the neural representation of a configuration

## Second win: Deep Learning classifiers are non-linear classifiers

- A softmax classifier assigns classes  $y \in C$  based on inputs  $x \in \mathbb{R}^d$  via the probability:

$$p(y|x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^C \exp(W_c \cdot x)}$$

- We train the weight matrix  $W \in \mathbb{R}^{C \times d}$  to minimize the neg. log loss :  $\sum_i -\log p(y_i|x_i)$
- Traditional ML classifiers (including Naïve Bayes, SVMs, logistic regression and softmax classifier) are not very powerful classifiers: they only give linear decision boundaries



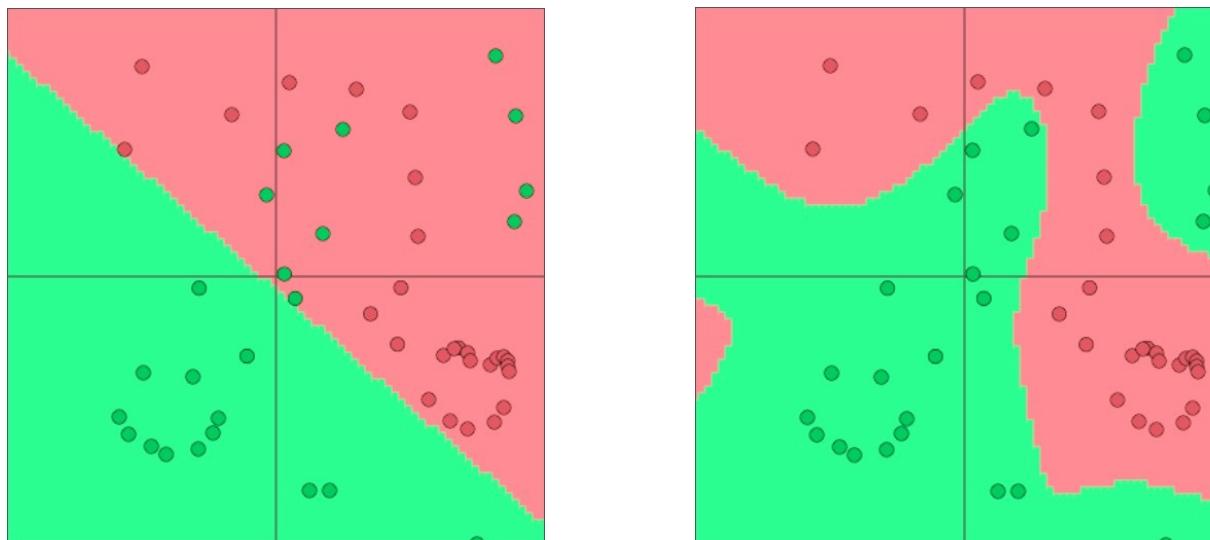
This can be quite limiting

→ Unhelpful when a problem is complex

Wouldn't it be cool to get these correct?

# Neural Networks are more powerful

- Neural networks can learn much more complex functions with nonlinear decision boundaries!
  - Non-linear in the original space, linear for the softmax at the top of the neural network



Visualizations with ConvNetJS by Andrej Karpathy!

<http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

# Simple feed-forward neural network multi-class classifier

Output layer  $y$   
 $y = \text{softmax}(Uh + b_2)$

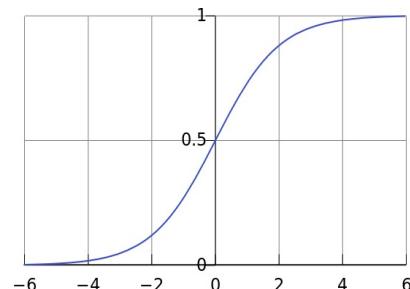
Hidden layer  $h$   
 $h = \text{ReLU}(Wx + b_1)$

Input layer  $x$

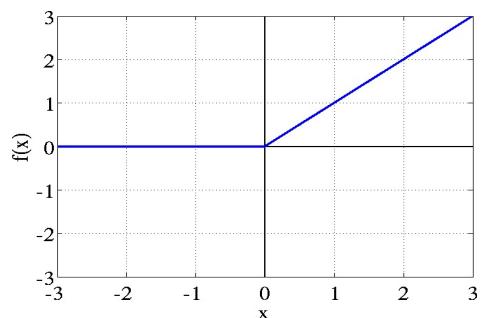
$x$  is result of lookup  
 $x_{(i, \dots, i+d)} = Le$   
lookup + concat

logistic = "sigmoid"

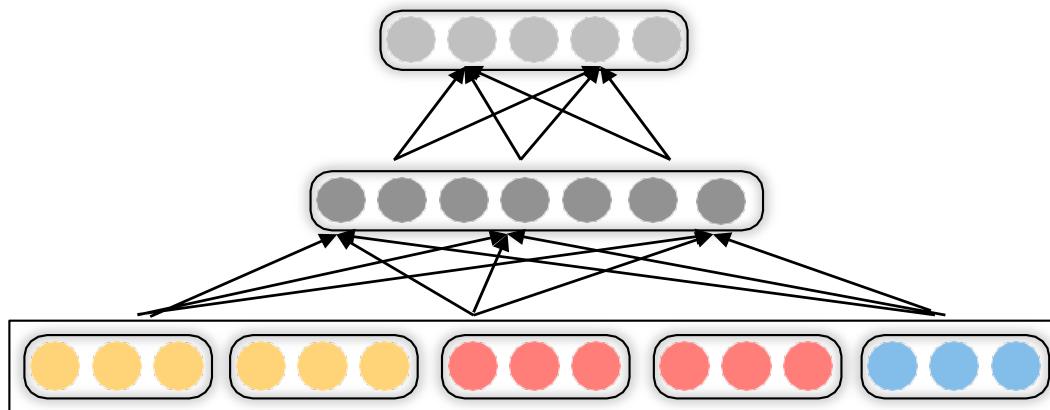
$$f(z) = \frac{1}{1 + \exp(-z)}.$$



ReLU = Rectified Linear Unit  
 $\text{rect}(z) = \max(z, 0)$



Softmax probabilities



Log loss (cross-entropy error) will be back-propagated to the embeddings

The hidden layer re-represents the input — it moves inputs around in an intermediate layer vector space—so it can be easily classified with a (linear) softmax

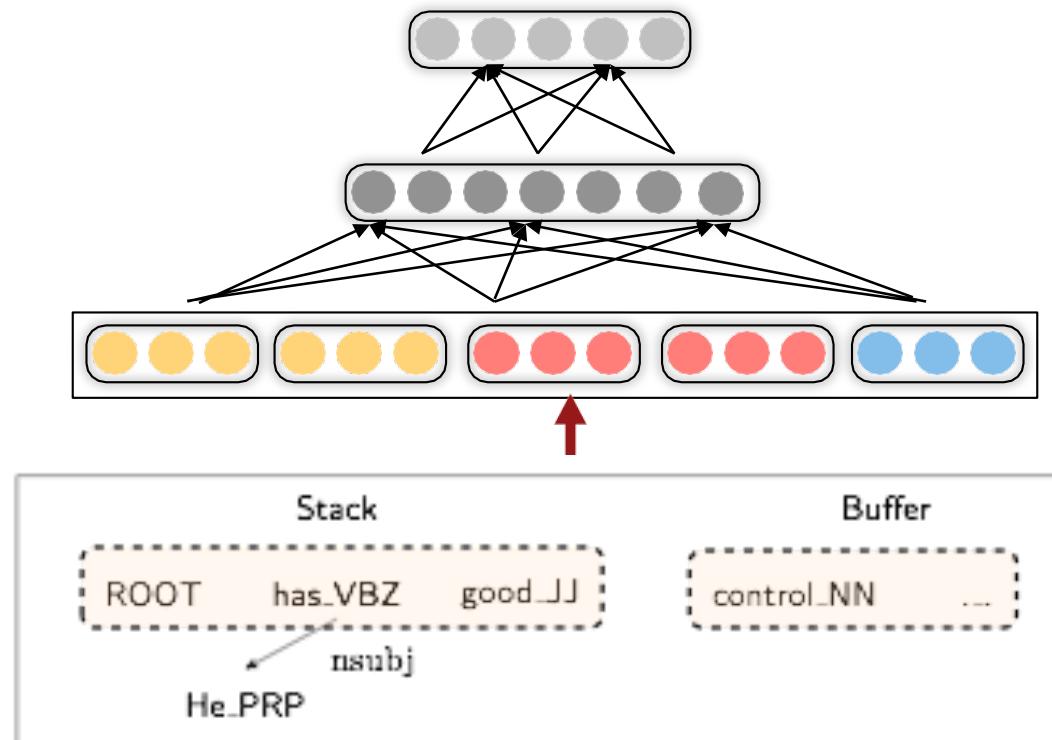
# Neural Dependency Parser Model Architecture

Output layer  $y$   
 $y = \text{softmax}(Uh + b_2)$

Hidden layer  $h$   
 $h = \text{ReLU}(Wx + b_1)$

Input layer  $x$   
lookup + concat

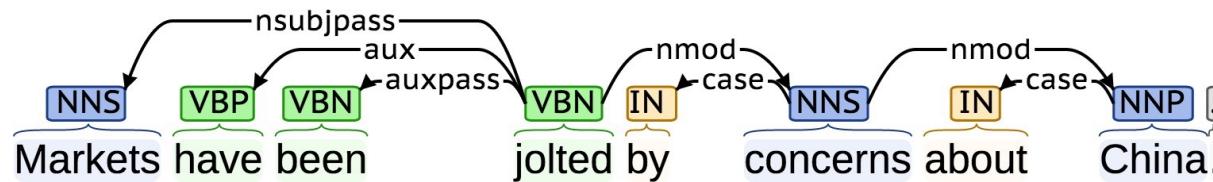
Softmax probabilities



cross-entropy error will be  
back-propagated to the  
embeddings

# Dependency parsing for sentence structure

- Neural networks can accurately determine the structure of sentences, supporting interpretation



- Chen and Manning (2014) was the first simple, successful neural dependency parser
- The dense representations (and non-linear classifier) let it outperform other greedy parsers in both accuracy and speed

# Further developments in transition-based neural dependency parsing

This work was further developed and improved by others, including in particular at Google

- Bigger, deeper networks with better tuned hyperparameters
  - Beam search
  - Global, conditional random field (CRF)-style inference over the decision sequence
- Leading to SyntaxNet and the Parsey McParseFace model (2016):  
“The World’s Most Accurate Parser”

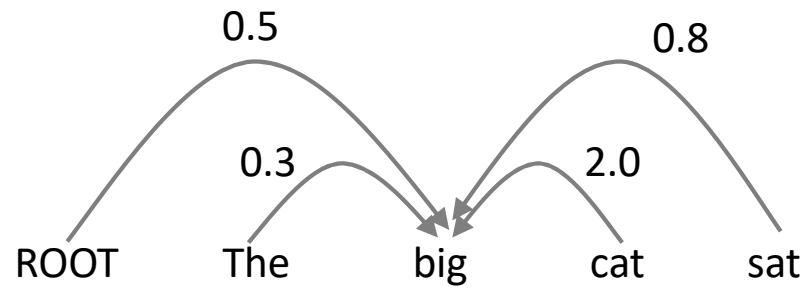
<https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html>



Method	UAS	LAS (PTB WSJ SD 3.3)
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79

# Graph-based dependency parsers

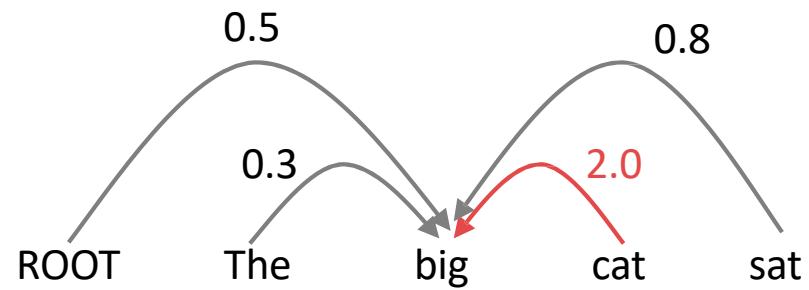
- Compute a score for every possible dependency for each word
  - Doing this well requires good “contextual” representations of each word token,



e.g., picking the head for “big”

# Graph-based dependency parsers

- Compute a score for every possible dependency for each word
  - Doing this well requires good “contextual” representations of each word token
  - And repeat the same process for each other word



e.g., picking the head for “big”

# A Neural graph-based dependency parser

[Dozat and Manning 2017; Dozat, Qi, and Manning 2017]

- This paper revived interest in graph-based dependency parsing in a neural world
  - Designed a biaffine scoring model for neural dependency parsing
    - Also crucially uses a neural sequence model
  - Really great results!
  - **But slower than the simple neural transition-based parsers**
    - There are  $n^2$  possible dependencies in a sentence of length  $n$



Method	UAS	LAS (PTB WSJ SD 3.3)
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
<b>Dozat &amp; Manning 2017</b>	<b>95.74</b>	<b>94.08</b>

**Thank you!**