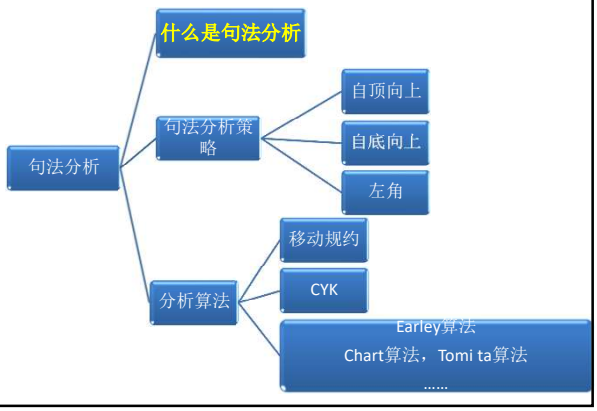
句法分析

内容提要



什么是句法分析

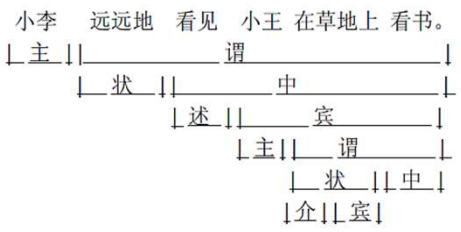
- 句法分析（Parsing）和句法分析器（Parser)
- 句法分析是从单词串得到句法结构的过程；
- 不同的语法形式，对应的句法分析算法也不尽相同；
- 句法分析包括：短语结构树和依存句法树

上下文无关文法（CFG）

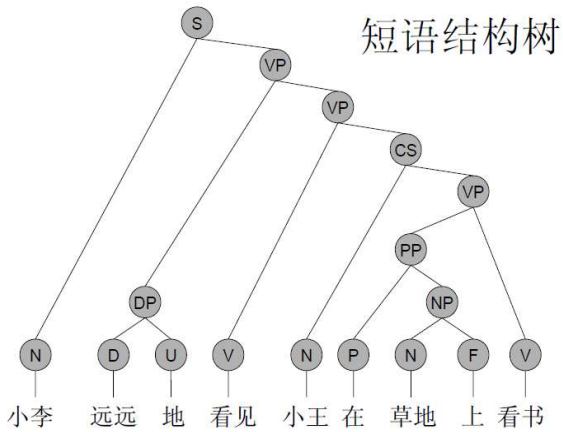
- 在计算机科学中，若一个形式文法G = (V, Σ, P, S) 的产生式规则都取如下的形式：N->w，则谓之。其中 N∈ V ，w∈(V∪Σ)* 。
  - V：非终结符号或变量的有限集合。
  - Σ是“终结符”的有限集合，
  - S是开始变量
  - F是从V到(V∪ Σ)*的关系,
- 上下文无关文法：因为字符 N 总可以被字串 w 自由替换，而无需考虑字符 V 出现的上下文。

词语到句子的组合关系

句法结构分析（syntactic structure parsing），作用是识别出句子中的短语结构以及短语之间的层次句法关系。



短语结构树



短语结构树

- G=<Vn,Vt,S,P>
- Vn={S,NP VP……}
- Vt={小李，远远，地，小王，在草地 上 看 书}
- P：S->N VP
  VP->DP VP
  DP->D U
  ……

词语之间的依赖关系

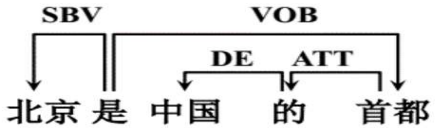
- **通过分析语言单位内成分之间的依存关系揭示其句法结构。直观来讲，依存句法分析识别句子中的“主谓宾”、“定状补”这些语法成分，并分析各成分之间的关系。**
- **现代依存语法(dependency grammar)理论的创立者是法国语言学家吕西安·泰尼埃：《结构句法概要》(1953)**

http://ictclas.nlpir.org/nlpir/

http://ltp.ai/demo.html

词语之间的依赖关系

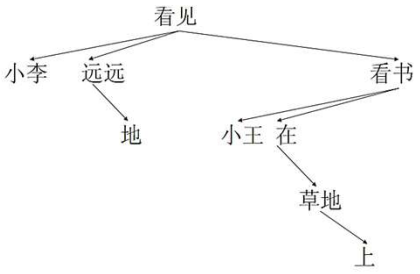
- 在依存语法理论中，“ 依存”就是指词与词之间支配与被支配的关系，这种关系不是对等的，而是有方向的。处于支配地位的成分称为支配者(governor, regent, head)，而处于被支配地位的成分称为从属者(modifier, subordinate. dependency)。



关系类型	Tag	Description	Example
主谓关系	SBV	subject-verb	我送她一束花 (我 <-- 送)
动宾关系	VOB	直接宾语，verb-object	我送她一束花 (送 --> 花)
间宾关系	IOB	间接宾语，indirect-object	我送她一束花 (送 --> 她)
前置宾语	FOB	前置宾语，fronting-object	他什么书都读 (书 <-- 读)
兼语	DBL	double	他请我吃饭 (请 --> 我)
定中关系	ATT	attribute	红苹果 (红 <-- 苹果)
状中结构	ADV	adverbial	非常美丽 (非常 <-- 美丽)
动补结构	CMP	complement	做完了作业 (做 --> 完)
并列关系	COO	coordinate	大山和大海 (大山 --> 大海)
介宾关系	POB	preposition-object	在贸易区内 (在 --> 内)
左附加关系	LAD	left adjunct	大山和大海 (和 <-- 大海)
右附加关系	RAD	right adjunct	孩子们 (孩子 --> 们)
独立结构	IS	independent structure	两个单句在结构上彼此独立
核心关系	HED	head	指整个句子的核心

依存结构树

- 处于支配地位的成分称为支配者(governor, regent, head)，而处于被支配地位的成分称为从属者(modifier, subordinate, dependency)



依存句法分析

- 1970年计算语言学家J. Robinson在论文《依存结构和转换规则》中提出了依存语法的4条公理:
  - **(1)一个句子只有一个独立的成分;**
  - **(2)句子的其他成分都从属于某一成分;**
  - **(3)任何一成分都不能依存于两个或多个成分;**
  - **(4)如果成分A直接从属于成分B，而成分C在句子中位于A和B之间，那么，成分C或者从属于A，或者从属于B，或者从属于A和B之间的某一成分。**

依存句法分析

- 依存语法中根据依存成分与中心语或姐妹成分在语序上的关系，可以分为符合投射性原则和 违反投射性原则两类。
- 在依存层级中如果每个依存成分与其中心或姐妹成分相邻，那么该依存层 级就是符合投射性原则（Projectivity）

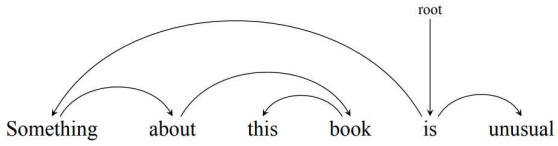


图 3.7　符合投射原则依存句法树样例

依存句法分析

- 如果依存成分的相邻成分使其与中心语分离，那么就是违反投射性原则的。这种现象在依存句 法中通常称为远距离依赖（Long-distance Dependencies）。

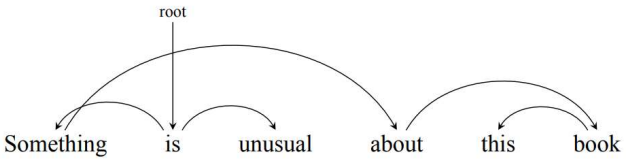


图 3.8　违反投射原则依存句法树样例

依存句法分析器

- **HanNLP，哈工大的LPT**
- **baidu-DDParser**
  - https://github.com/baidu/DDParser
  - https://blog.csdn.net/qq_27590277/article/details/107853326
- **南京大学的Dependency Viewer**
  - http://nlp.nju.edu.cn/tanggc/tools/DependencyViewer.html

句法的应用

- 情感分析
  1. 语义依赖分析：以依存句法分析为基础，判断情感词在句子中的语法成分。
  2. 句子极性计算：根据情感词汇及其语法成分、情感词与否定词之间的句法关系，构建规则，计算句子极性。
  3. 情感聚合：考虑篇章结构，对主观句的情感值进行加权求和，判断篇章情感倾向。主观句距离篇章首尾越近，权值越大。

http://nlp.stanford.edu:8080/sentiment/rntnDemo.html

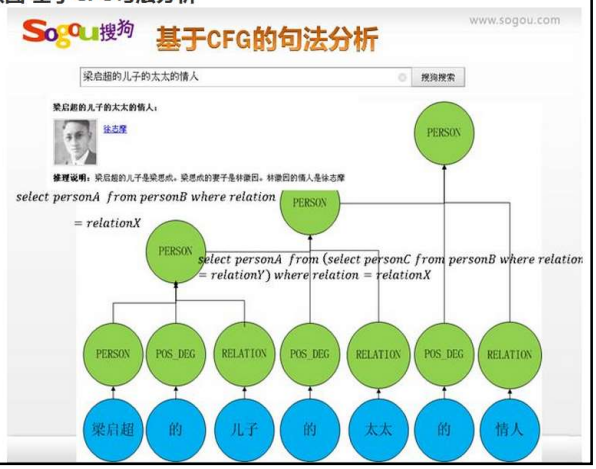
其中：句子极性计算依赖的规则构建基于论文 **Sentiment Analysis of Chinese Documents: From Sentence to Document Level, 2014**
Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank (EMNLP 2013)

句法的应用

- 事件抽取
  - 判断句子描述的事件，通过核心谓词和语法结构，进行判断。
  - 常用的语法关系包括：核心谓词的并列关系（COO）词；核心谓词的动宾关系（VOB）词；
  - 在处理长句子时，还需用到核心谓词的多级并列关系词（并列的并列……）。
  - 一个用于事件抽取的开源代码：使用句法依存分析抽取事实三元组

https://github.com/twjiang/fact_triple_extraction

第七张图 基于CFG句法分析



上下文无关语法

终结符 $\sum$（Terminal Symbols）：与语言中词汇对应的符号，用 u, v, w 等小写罗马字母表示；
例如：上海，walk

非终结符 N（Non-terminals）：对应词组等聚集类或概括性符号，用 A,B,C 等大写字母表示；
例如：NP（名词短语），VP（动词短语），N（名词），介词（P）

初始符 S（Start symbol）：语法中指定的初始符，通常用 S 表示；

规则 R（rules）：对应语法中的短语结构规则，从初始符号开始可以构造出合法句子的规则集合，在上下文无关语法中，一个规则的左边是一个单独的非终结符，右边是一个或者多个终结符或非终结符组成的有序序列 $(\sum \cup N)*$。用 α, β, γ 等小写希腊字母表示 $(\sum \cup N)*$。
例如：S→ NP VP，NP→ Det Adj N，Det → the

句法结构的歧义消解

- 我是县长。
- 我是县长派来的。
- 咬死了猎人的狗跑了。
- 就是这条狼咬死了猎人的狗。
- 小王和小李的妹妹结婚了。
- 小王和小李的妹妹都结婚了。

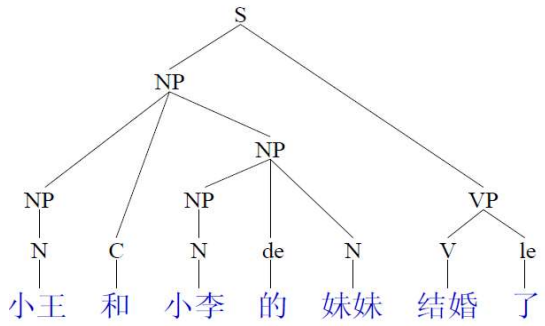
例子－语法

- 小王和小李的妹妹结婚了

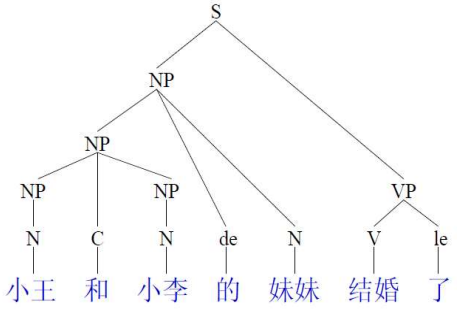
规则：
S → NP VP
NP → NP C NP
NP → N
NP → NP de N
VP → V le

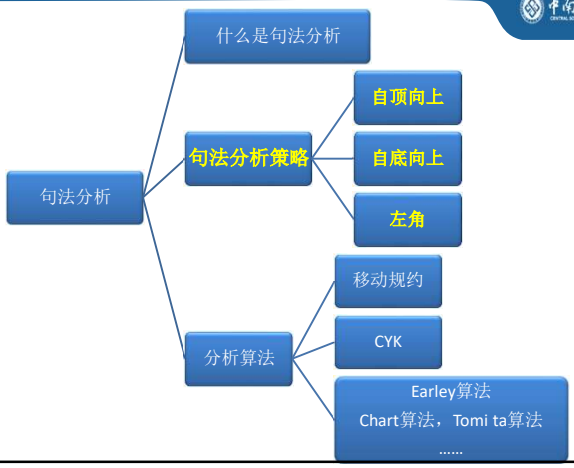
词典：
小王：N
小李：N
和：C
妹妹：N
结婚：V
了：le
的：de

例子－分析结果之一



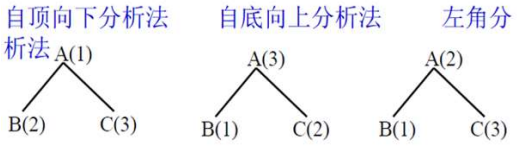
例子－分析结果之二





自顶向下和自底向上分析法 (1)

- 句法树的构造过程
  - 所谓自顶向下分析法也就是先构造句法树的根结点，再逐步向下扩展，直到叶结点；
  - 所谓自底向上分析法也就是先构造句法树的叶结点，再逐步向上合并，直到根结点。
  - 左角分析：自底向上分析法与自顶向下分析法结合



自顶向下和自底向上分析法 (2)

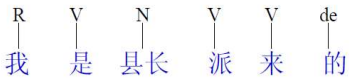
- 自顶向下的方法又称为基于预测的方法，也就是说，这种方法是先产生对后面将要出现的成分的预期，然后再通过逐步吃进待分析的字符串来验证预期。如果预期得到了证明，就说明待分析的字符串可以被分析为所预期的句法结构。
- 如果某一个环节上预期出了差错，那就要用另外的预期来替换(即回溯)。如果所有环节上所有可能的预期都被吃进的待分析字符串所“反驳”，那就说明待分析的字符串不可能是个合法的句子，分析失败。

自顶向下和自底向上分析法 (2)

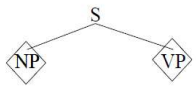
- 自底向上的方法也叫基于归约的方法。就是说，这种方法是先逐步吃进待分析字符串，把它们从局部到整体层层归约为可能的成分。如果整个待分析字符串被归约为开始符号 S，那么分析成功。
- 如果在某个局部证明不可能有任何从这里把整个待分析字符串归约为句子的方案，那么就需要回溯。如果某一个环节上预期出了差错，那就要用另外的预期来替换(即回溯)。如果所有环节上所有可能的预期都被吃进的待分析字符串所“反驳”，那就说明待分析的字符串不可能是个合法的句子，分析失败。

自顶向下分析法—示例 (1)

查词典

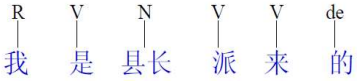


自顶向下分析法—示例 (2)

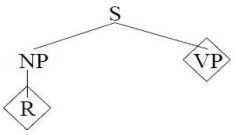


使用规则：

S → NP VP



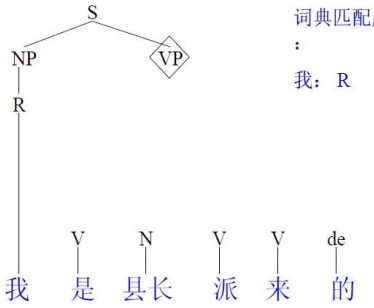
自顶向下分析法—示例 (3)



使用规则：

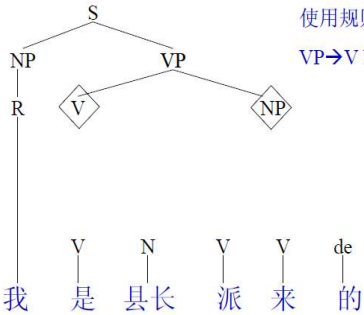
NP→R

自顶向下分析法－示例 (4)



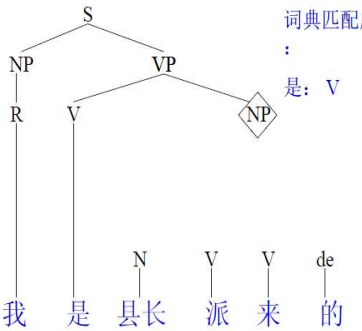
词典匹配成功：
我：R

自顶向下分析法－示例 (5)



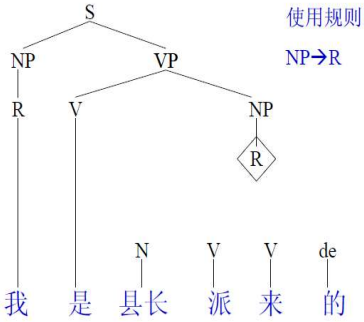
使用规则：
VP→V NP

自顶向下分析法－示例 (6)



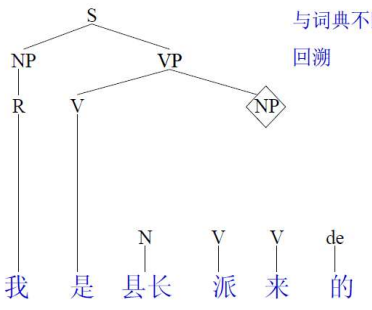
词典匹配成功：
是：V

自顶向下分析法－示例 (7)



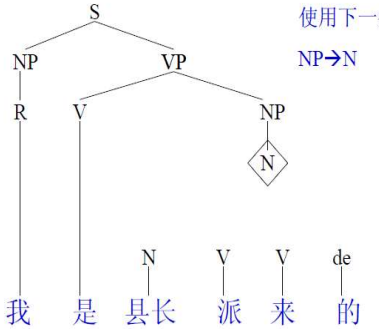
使用规则：
NP→R

自顶向下分析法－示例 (8)



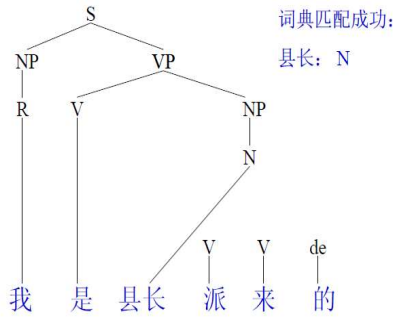
与词典不匹配
回溯

自顶向下分析法－示例 (9)

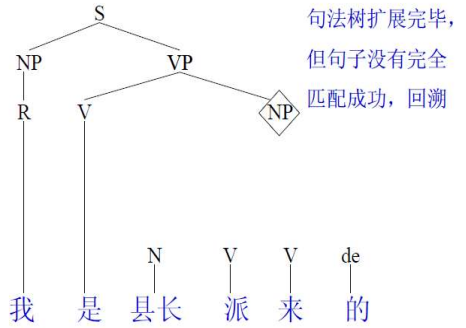


使用下一条规则：
NP→N

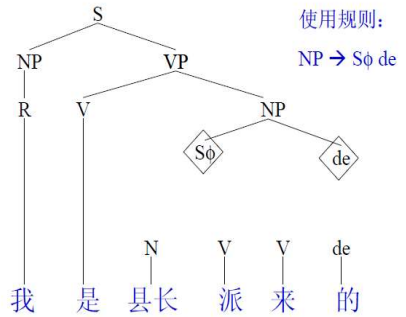
自顶向下分析法－示例 (10)



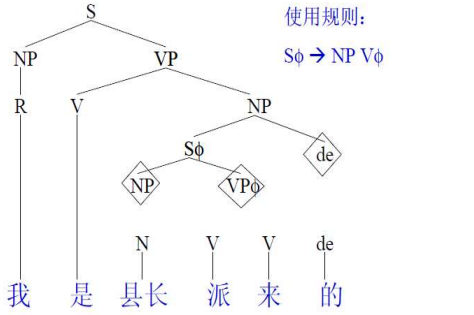
自顶向下分析法－示例 (11)



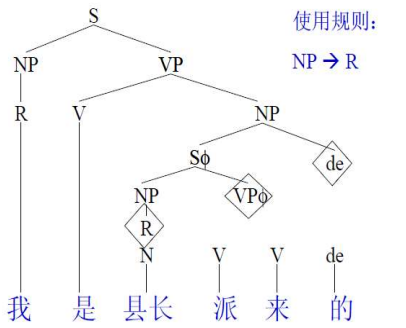
自顶向下分析法－示例 (12)



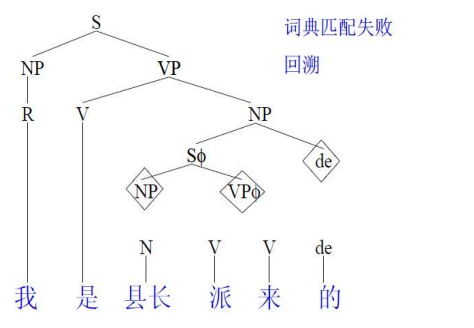
自顶向下分析法－示例 (13)



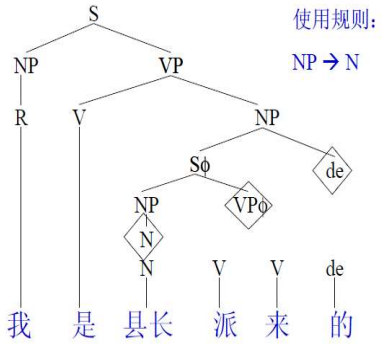
自顶向下分析法－示例 (14)



自顶向下分析法－示例 (15)



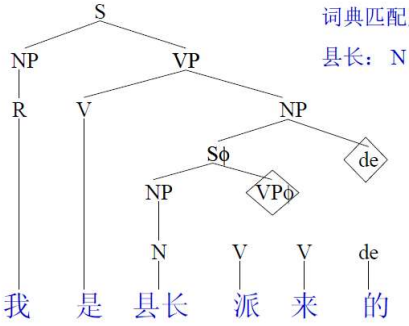
自顶向下分析法－示例 (16)



使用规则：

NP → N

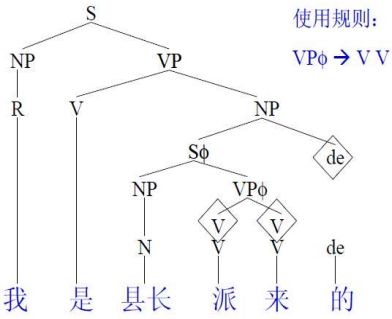
自顶向下分析法－示例 (17)



词典匹配成功：

县长： N

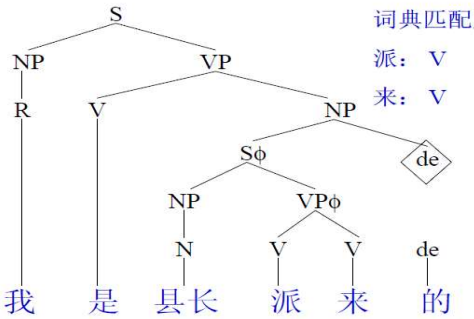
自顶向下分析法－示例 (18)



使用规则：

VPϕ → V V

自顶向下分析法－示例 (19)

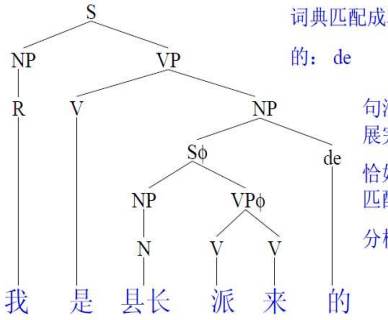


词典匹配成功：

派： V

来： V

自顶向下分析法－示例 (20)



词典匹配成功：

的： de

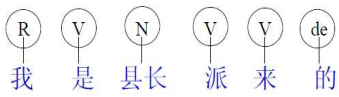
句法树扩展完毕

恰好句子匹配完成

分析成功

自底向上分析法－示例 (1)

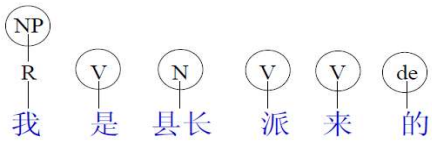
查词典



自底向上分析法－示例 (2)

使用规则：

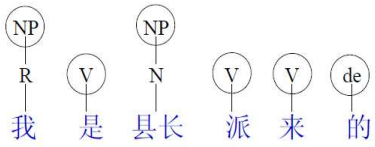
NP→R



自底向上分析法－示例 (3)

使用规则：

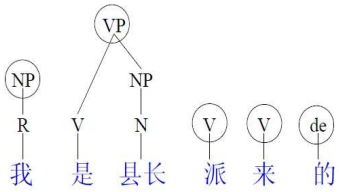
NP→N



自底向上分析法－示例 (4)

使用规则：

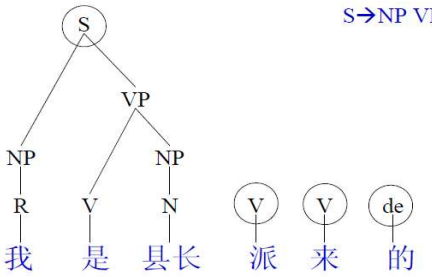
VP→V NP



自底向上分析法－示例 (5)

使用规则：

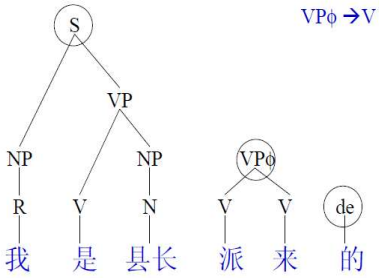
S→NP VP



自底向上分析法－示例 (6)

使用规则：

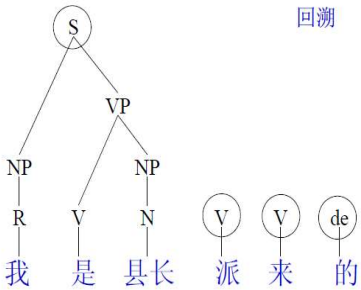
VPϕ →V V



自底向上分析法－示例 (7)

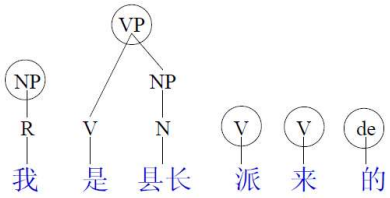
无规则可用

回溯



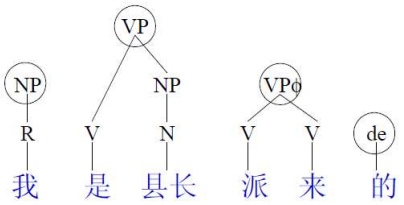
自底向上分析法－示例 (8)

无规则可用，
回溯



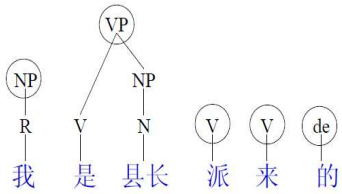
自底向上分析法－示例 (9)

使用规则：
VPϕ →V V



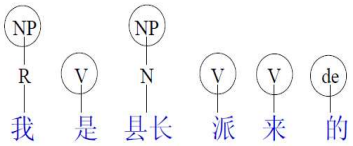
自底向上分析法－示例 (10)

无规则可用，
回溯



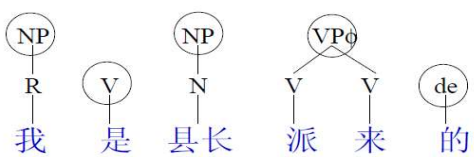
自底向上分析法－示例 (11)

无规则可用，
回溯



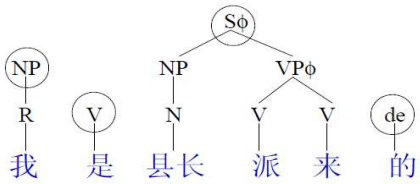
自底向上分析法－示例 (12)

使用规则：
VPϕ →V V



自底向上分析法－示例 (13)

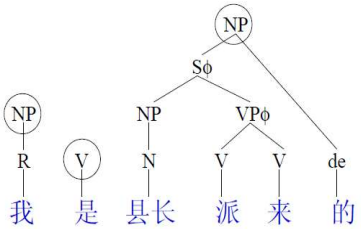
使用规则：
Sϕ →NP VPϕ



自底向上分析法－示例 (14)

使用规则：

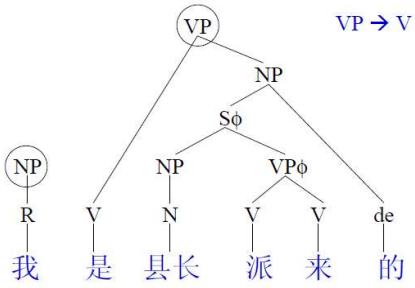
NP → Sϕ de



自底向上分析法－示例 (15)

使用规则：

VP → V NP

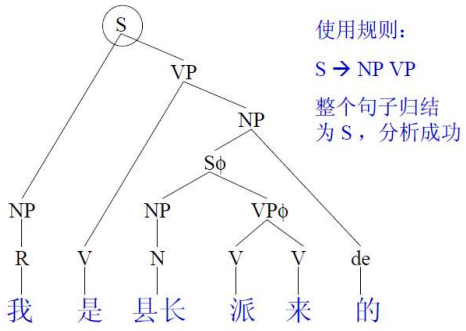


自底向上分析法－示例 (16)

使用规则：

S → NP VP

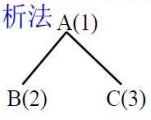
整个句子归结为 S，分析成功



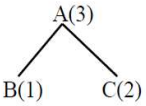
左角分析法－概述

- 左角分析法是一种自顶向下和自底向上相结合的方法
- 所谓“左角 (Left Corner)”是指任何一个句法子树中左下角的那个符号
- 比较：

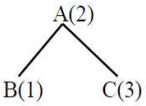
自顶向下分析法



自底向上分析法

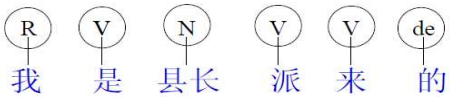


左角分析法



左角分析法－示例 (1)

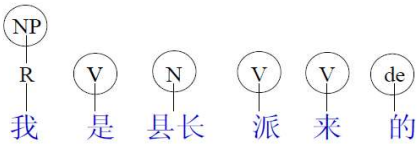
查词典



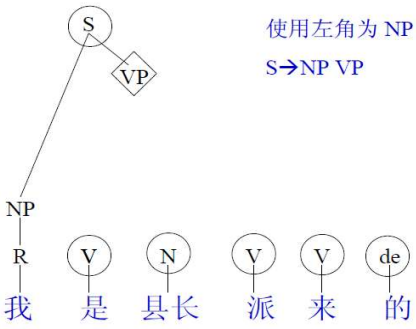
左角分析法－示例 (2)

使用左角为 R 的规则：

NP→R



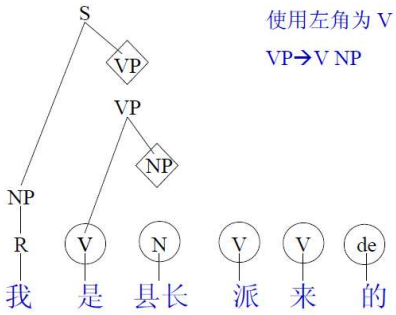
左角分析法－示例 (3)



使用左角为 NP 的规则：

S→NP VP

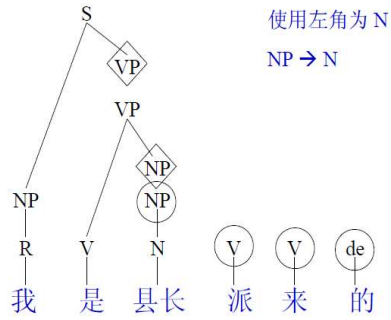
左角分析法－示例 (4)



使用左角为 V 的规则：

VP→V NP

左角分析法－示例 (5)



使用左角为 N 规则：

NP → N

不同分析策略的比较

- 不同的分析策略在面对不同的语法规则和句子的时候，各有优势
- 对于某一种具体的自然语言，可能总体上会存在一种比较好的分析策略

上下文无关语法的分析算法

常见的上下文无关语法的句法分析算法：

1. CYK 算法；
2. 移进－归约算法；
3. Marcus 确定性分析算法；
4. Earley 算法；
5. Tomita 算法（ GLR 算法、富田算法）；
6. Chart 算法（图分析算法、线图分析算法）；

移进-归约算法：概述

- 移进－归约算法： Shift-Reduce Algorithm
- 移进－归约算法类似于下推自动机的 LR 分析算法
- 移进－归约算法的基本数据结构是堆栈
- 移进－归约算法的四种操作：
  - 移进：从句子左端将一个终结符移到栈顶
  - 归约：根据规则，将栈顶的若干个符号替换成一个符号
  - 接受：句子中所有词语都已移进到栈中，且栈中只剩下一个符号 S，分析成功，结束
  - 拒绝：句子中所有词语都已移进栈中，栈中并非只有一个符号 S，也无法进行任何归约操作，分析失败，结束

移进－归约算法：冲突

- 移进－归约算法中有两种形式的冲突：
  - 移进－归约冲突：既可以移进，又可以归约
  - 归约－归约冲突：可以使用不同的规则归约
- 冲突解决方法：回溯
- 回溯导致的问题：
  - 回溯策略：对于互相冲突的各项操作，给出一个选择顺序
  - 断点信息：除了在堆栈中除了保存非终结符外，还需要保存断点信息，使得回溯到该断点时，能够恢复堆栈的原貌，并知道还可以有哪些可选的操作

移进－归约算法：示例 (1)

- 回溯策略：
  - 移进－归约冲突：先归约，后移进
  - 归约－归约冲突：规则事先排序，先执行排在前面的规则
- 断点信息：
  - 当前规则：标记当前归约操作所使用的规则序号
  - 候选规则：记录在当前位置还有哪些规则没有使用（由于这里规则是排序的，所以这一条可以省略）
  - 被替换结点：归约时被替换的结点，以便回溯时恢复

移进－归约算法：示例 (2)

- 给规则排序并加上编号：

规则：

(7) NP → R
(8) NP → N
(9) NP → Sϕ de
(10) VPϕ → V V
(11) VP → V NP
(12) Sϕ → NP VPϕ
(13) S → NP VP

词典：

(1) R→ 我
(2) N → 县长
(3) V → 是
(4) V → 派
(5) V → 来
(6) de → 的

移进－归约算法：示例 (3)

步骤	栈	输入	操作	规则
1	#	我 是 县长 派 来 的	移进	
2	# 我	是 县长 派 来 的	归约	(1) R→ 我
3	# R (1)	是 县长 派 来 的	归约	(7) NP→R
4	# NP (7)	是 县长 派 来 的	移进	
5	# NP (7) 是	县长 派 来 的	归约	(3) V→ 是
6	# NP (7) V (3)	县长 派 来 的	移进	
7	# NP (7) V (3) 县长	派 来 的	归约	(2) N→ 县长
8	# NP (7) V (3) N (2)	派 来 的	归约	(8) NP→N
9	# NP (7) V (3) NP (8)	派 来 的	归约	(11) VP→V NP
10	# NP (7) VP (11)	派 来 的	归约	(13) S→NP VP
11	# S (13)	派 来 的	移进	

移进－归约算法：示例 (4)

步骤	栈	输入	操作	规则
12	# S (13) 派	来 的	归约	(4) V→ 派
13	# S (13) V(4)	来 的	移进	
14	# S (13) V(4) 来	的	归约	(5) V→ 来
15	# S (13) V(4) V(5)	的	归约	(10) VPϕ → V V
16	# S (13) VPϕ (10)	的	移进	
17	# S (13) VPϕ (10) 的		归约	(6) de→ 的
18	# S (13) VPϕ (10) de(6)		回溯	
19	# S (13) VPϕ (10) 的		回溯	
20	# S (13) VPϕ (10)	的	回溯	
21	# S (13) V(4) V (5)	的	回溯	
22	# S (13) V(4) 来	的	回溯	

移进－归约算法：示例 (5)

步骤	栈	输入	操作	规则
23	# S (13) V(4)	来 的	回溯	
24	# S (13) 派	来 的	回溯	
25	# S (13)	派 来 的	回溯	
26	# NP (7) VP (11)	派 来 的	移进	(13) S→NP VP
27	# NP (7) VP (11) 派	来 的	归约	(4) V→ 派
28	# NP (7) VP (11) V(4)	来 的	移进	
29	# NP (7) VP (11) V(4) 来	的	归约	(5) V→ 来
30	# NP (7) VP (11) V(4) V(5)	的	归约	(10) VPϕ → V V
31	# NP (7) VP (11) VPϕ (10)	的	移进	
32	# NP (7) VP (11) VPϕ (10) 的		归约	(6) de→ 的
33	# NP (7) VP (11) VPϕ (10) de(6)		回溯	

移进－归约算法：示例 (6)

步骤	栈	输入	操作	规则
34	# NP (7) VP (11) VPϕ (10) 的		回溯	
35	# NP (7) VP (11) VPϕ (10)	的	回溯	
36	# NP (7) VP (11) V(4) V(5)	的	回溯	
37	# NP (7) VP (11) V(4) 来	的	回溯	
38	# NP (7) VP (11) V(4)	来 的	回溯	
39	# NP (7) VP (11) 派	来 的	回溯	
40	# NP (7) VP (11)	派 来 的	回溯	
41	# NP (7) VP (11)	派 来 的	回溯	
42	# NP (7) V (3) NP (8)	派 来 的	移进	
43	# NP (7) V (3) NP (8) 派	来 的	归约	(4) V→ 派
44	# NP (7) V (3) NP (8) V(4)	来 的	移进	

移进－归约算法：示例 (7)

步骤	栈	输入	操作	规则
45	# NP (7) V (3) NP (8) V(4) 来	的	归约	(5) V→ 来
46	# NP (7) V (3) NP (8) V(4) V(5)	的	归约	(10) VPϕ → V V
47	# NP (7) V (3) NP (8) VPϕ (10)	的	归约	(12) Sϕ → NP VPϕ
48	# NP (7) V (3) Sϕ (12)	的	移进	
49	# NP (7) V (3) Sϕ (12) 的		归约	(6) de→ 的
50	# NP (7) V (3) Sϕ (12) de (6)		归约	(9) NP → Sϕ de
51	# NP (7) V (3) NP (9)		归约	(11) VP→V NP
52	# NP (7) VP (11)		归约	(13) S→NP VP
53	# S (13)		接受	

说明：为简洁起见，这个例子中没有给出堆栈中被替换结点信息，但必须注意到这些信息是必需的，否则归约操作将无法回溯。

移进－归约算法：特点

- 移进一归约算法是一种自底向上的分析算法
- 为了得到所有可能的分析结果，可以在每次分析成功时都强制性回溯，直到分析失败
- 采用回溯算法将导致大量的冗余操作，效率非常低

移进－归约算法的改进

- 如果在出现冲突(移进一归约冲突和归约一归约冲突)时能够减少错误的判断，将大大提高分析的效率
- 引入规则:通过规则，给出在特定条件 (栈顶若干个符号和待移进的单词) 应该采取的动作
- 引入上下文:考虑更多的栈顶元素和更多的待移进单词来写规则
- 引入缓冲区 (Marcus 算法) : 是一种确定性的算法，没有回溯，但通过引入缓冲区，可以延迟作出决定的时间

CYK算法

- CYK算法：Cocke-Younger-Kasami算法
- CYK算法是一种并行算法，不需要回溯；
- CYK算法建立在Chomsky范式的基础上
  - Chomsky范式的规则只有两种形式：A→BC A→x 这里A,B,C是非终结符，x是终结符
  - 由于后一种形式实际上就是词典信息，在句法分析之前已经进行了替换，所以在分析中我们只考虑形如A→BC形式的规则
  - 由于任何一个上下文无关语法都可以转化成符合Chomsky范式的语法，因此CYK算法可以应用于任何一个上下文无关语法

CYK算法一数据结构 1

6	S					
5		VP				
4			NP			
3	S		Sϕ			
2		VP		VPϕ		
1	NP,R	V	NP,N	V	V	de
	1	2	3	4	5	6
	我	是	县长	派	来	的

CYK算法—数据结构 2

- 一个二维矩阵：{ P(i , j) }
  - 每一个元素P(i , j)对应于输入句子中某一个跨度（Span）上所有可能形成的短语的非终结符的集合
  - 横坐标 i：该跨度左侧第一个词的位置
  - 纵坐标 j：该跨度包含的词数
- 上图中P(3,1)={NP,N}表示“县长”可以归约成N和NP，P(3,3)={Sϕ}表示“县长 派 来”可以规约成Sϕ

CYK算法:算法描述

1. 对i=1…n，j=1（填写第一行，长度为1）
   对于每一条规则A→Wi，
   将非终结符A加入集合P(i,j)；
2. 对j=2…n（填写其他各行，长度为j）
   对i=1…n-j+1（对于所有起点i）
   对k=1…j-1（对于所有左子结点长度k）
   对每一条规则A→BC，
   如果B ∈ P(i,k)且C∈P(i+k,j-k)
   那么将非终结符A加入集合P(i,j)
3. 如果S∈P(1,n)，那么分析成功，否则分析失败

CYK算法：特点

- 本质上是一种自底向上分析法；
- 采用广度优先的搜索策略；
- 采用并行算法，不需要回溯，没有冗余的操作；
- 时间复杂度$O(n^3)$；
- 由于采用广度优先搜索，在歧义较多时，必须分析到最后才知道结果，无法采用启发式策略进行改进。

概率分布的上下文无关语法
（Probabilistic Context-Free Grammar）

- PCFG （Probabilistic Context-Free Grammar）
  - 除了常规的语法规则以外，我们还对每一条规则赋予了一个概率。
  - 对于每一棵生成的语法树，我们将其中所以规则的概率的乘积作为语法树的出现概率。
  - 当得多棵语法树时，我们可以分别计算每颗语法树的概率p(t)，出现概率最大的那颗语法树就是我们希望得到的结果，即arg max p(t)

概率分布的上下文无关语法
（Probabilistic Context-Free Grammar）

- **PCFG 即 Probabilistic CFG，也就是基于概率的短语结构分析**。也就是在 G = (N,∑,S,R) 的基础上加一个 P 表示概率，变成 G= (N,∑,S,R,P) 约束如下：

$$\sum_{\alpha} P(A->\alpha)=1$$

- 即：非终结符 A 的转换生成概率之和为 1.

$\alpha \in (\sum \cup N)*$ 为终结符和非终结符组成的有序序列集合,

概率分布的上下文无关语法
（Probabilistic Context-Free Grammar）

比如有一个文法概率内容如下：

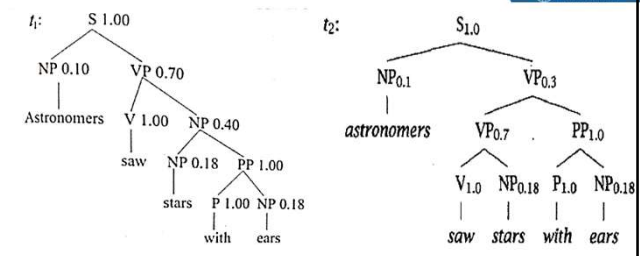
S -> NP VP , 1.00
NP -> NP PP, 0.40
VP -> VP PP , 0.30
PP -> P NP , 1.00
VP -> V NP , 0.70
NP -> astronomers , 0.10
NP -> saw, 0.04
V -> saw, 1.00
NP -> telescopes ,0.1
P -> with, 1.00
NP -> ears, 0.18
NP -> stars , 0.18

模型的假设

- 基于PCFG的句法分析模型，满足以下三个条件：
  - 位置不变性：子树的概率不依赖于该子树所管辖的单词在句子中的位置
  - 上下文无关性：子树的概率不依赖于子树控制范围以外的单词
  - 祖先无关性：子树的概率不依赖于推导出子树的祖先节点

- 基于上述独立性假设，一个特定句法树 T 的概率定义为该句法树 T 中用来得到句子 W 所使用的 m 个规则的概率乘积：

$$P(T) = \prod_{i=1}^{m} P(A_i \rightarrow \alpha)$$



$$P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18$$
$$= 0.000\ 907\ 2$$
$$P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18$$
$$= 0.000\ 680\ 4$$
$$P(w_{15}) = P(t_1) + P(t_2) = 0.001\ 587\ 6$$

概率分布的上下文无关语法

（Probabilistic Context-Free Grammar）

- 与HMM类似，PCFG也有三个基本问题：
  - 给定一个句子W＝w1w2…wn和文法G，如何快速计算概率P(W|G)
  - 给定一个句子W＝w1w2…wn和文法G，如何选择该句子的最佳结构？即选择句法结构树t使其具有最大概率
  - 给定PCFG G和句子W＝w1w2…wn，如何调节G的概率参数，使句子的概率最大

概率分布的上下文无关语法

（Probabilistic Context-Free Grammar）

最佳树结构求解是指对于给定句子 w = {wl,w2.·,wn} 和 PCFG 文法 G，求解该句子的最佳树结构，即如何选择句法结构树使得其概率最大:

$$\hat{t} = \arg\max_{t \in \mathcal{T}_G(W)} P(t|W,G)$$

$T_G(W)$ 表示句子 W 所有符合文法 G 的法树。该问题可以通过利用基于概率的 CYK 算法进行

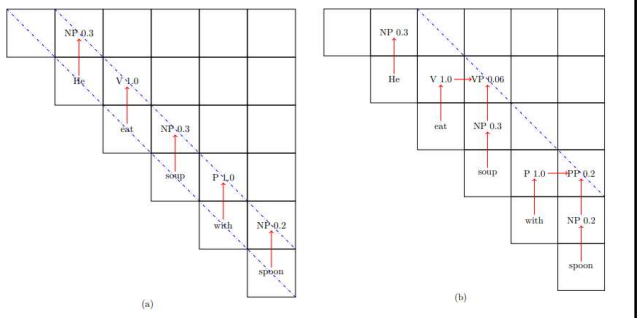
概率上下文无关语法问题

- Penn Treebank通过手工的方法已经提供了一个非常大的语料数据集，
- 我们的任务就是从语料库中训练出PCFG所需要的参数。
  - 1）统计出语料库中所有的N与Σ；
  - 2）利用语料库中的所有规则作为R；
  - 3）针对每个规则A -> B，从语料库中估算p(x) = p(A -> B) / p(A)

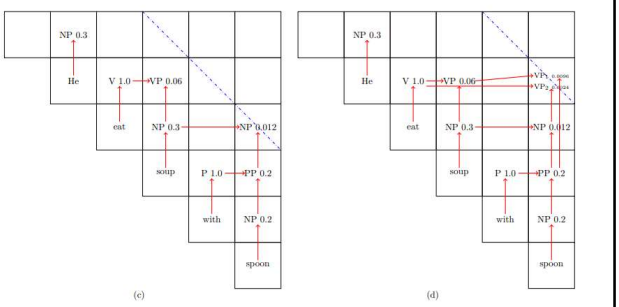
概率上下文无关语法问题

- 针对句子长度为 n 的句子，概率 CYK 算法同样也使用(n + 1) x (n + 1) 矩阵 T 的上三角形部分进行存储。
- T_{ij}，表示输入句子中横跨在位置i到j之间的单词的组成成分，包含由文法 G 的非终结符构成的集合以及以该非终结符为根的句法树的概率。
- 为了方便起见这里我们使用 $T_{ij,A}$ 表示矩阵 T_{ij} 保存的非终结符集合中 A 的概率。

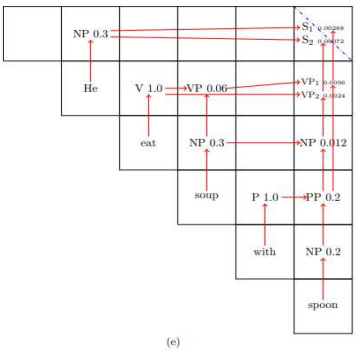
非终结符号集合：$N = \{S, P, V, NP, PP, VP\}$
终结符号集合：$\sum = \{$ eat, he, soup, spoon, with$\}$
规则集合：$R = \{$ (1) S → NP VP 1.0;　(2) VP → VP PP 0.8;　(3) VP → V NP 0.2;
(4) NP → NP PP 0.2;　(5) PP → P NP 1.0 (6) NP → He 0.3 ;



非终结符号集合：$N = \{S, P, V, NP, PP, VP\}$
终结符号集合：$\sum = \{$ eat, he, soup, spoon, with$\}$
规则集合：$R = \{$ (1) S → NP VP 1.0;　(2) VP → VP PP 0.8;　(3) VP → V NP 0.2;
(4) NP → NP PP 0.2;　(5) PP → P NP 1.0 (6) NP → He 0.3 ;



非终结符号集合：$N = \{S, P, V, NP, PP, VP\}$
终结符号集合：$\sum = \{$ eat, he, soup, spoon, with$\}$
规则集合：$R = \{$ (1) S → NP VP 1.0;　(2) VP → VP PP 0.8;　(3) VP → V NP 0.2;
(4) NP → NP PP 0.2;　(5) PP → P NP 1.0 (6) NP → He 0.3 ;



代码 3.4: 概率 CYK 句法分析算法

输入: 语法信息 G 和句子 $w_1 w_2 \cdots w_n$
输出: 句法树矩阵 T

```
// 初始化
for i = 1 to n do
    T_ii = w_i ;                    // 主对角线上依次放入单词 w_i;
    foreach A|A → w_i do
        T_(i-1)i = T_(i-1)i ∪ (A, p), (A → w_i, p ∈ G) ; // 依次放入单词 w_i 的所有词性及其概率;
    end
end
// 平行于主对角线逐层计算
for d = 2 to n do
    for i = 0 to n − d do
        j = i+d;
        for k = i + 1 to j − 1 do
            if (A → BC, p) ∈ G and T_ik,B > 0 and T_kj,C > 0 then
                if T_ij,A < p × T_ik,B × T_kj,C then
                    T_ij = T_ij ∪ {A, p × T_ik,B × T_kj,C, (k, B, C)}; // (k,B,C) 用于保存推导路径;
                end
            end
        end
    end
end
return T
```

短语句法分析评价方法

目前使用比较广泛的句法分析器评价指标是PARSEVAL测度，三个基本的评测指标：

- 精度(precision)：句法分析结果中正确的短语个数所占的比例，即分析结果中与标准分析树（答案）中的短语相匹配的个数占分析结果中所有短语个数的比例，即：

$$P = \frac{\text{分析得到的正确的短语个数}}{\text{分析得到的所有的短语个数}} \times 100\%$$

短语句法分析评价方法

- 召回率(recall)：句法分析结果中正确的短语个数占标准分析树中全部短语个数的比例，即：

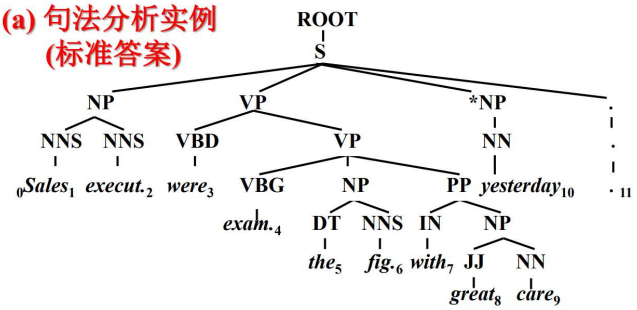
$$R=\frac{\text{分析得到的正确的短语个数}}{\text{标准树库中(答案)的短语个数}}\times 100\%$$

短语句法分析评价方法

例如：下面的图(a)为句子 “*Sales executives were examining the figures with great care yesterday.*” 的正确分析树(答案标准)。

短语句法分析评价方法

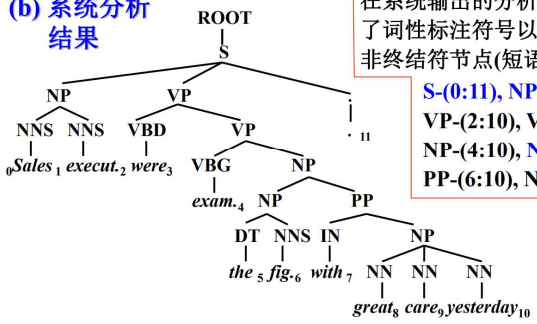
(a) 句法分析实例 (标准答案)



在标准答案树中，除了词性标注符号以外(即除了叶子节点和其直接父节点以外)的其他非终结符节点(短语)有：S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7:9), *NP-(9:10) 。

短语句法分析评价方法

(b) 系统分析结果



在系统输出的分析树中，除了词性标注符号以外的其他非终结符节点(短语)有：S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:10), NP-(4:6), PP-(6:10), NP-(7:10)。

短语句法分析评价方法

标准答案：S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7:9), *NP-(9:10)

系统结果：S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:10), NP-(4:6), PP-(6:10), NP-(7:10)

$$\text{Precision}=\frac{3}{8}\times 100\%=37.5\%$$

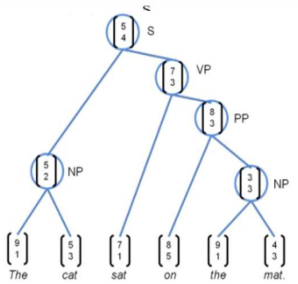
$$\text{Recall}=\frac{3}{8}\times 100\%=37.5\%$$

注：图(a)中加*号的一元的节点 (*NP) 在计算时应该被排除在外，但在这里也被包括进来了。

一些可免费使用的Parser

- Michael Collins ‘s Parser
  - http://people.csail.mit.edu/mcollins/code.html
  - English
- Dan Bikel ‘s Parser
  - http://www.cis.upenn.edu/~dbikel/software.html#stat-parser
  - English / Chinese / Arabic
- Stanford Parser
  - http://www-nlp.stanford.edu/software/lex-parser.shtml
  - English / Chinese / German
- David Chiang’s Parser
  - http://www.isi.edu/~chiang/
  - English / Chinese

基于深度学习的句法分析



- 深度学习
  - 每个单词或者短语都是一个向量
  - 神经网络用于向量的两两合并
  - Socher et al. 2011