中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

机器翻译原理与方法

第三讲 句法分析技术

刘群
中国科学院计算技术研究所
liuqun@ict.ac.cn

中国科学院计算技术研究所2011年秋季课程

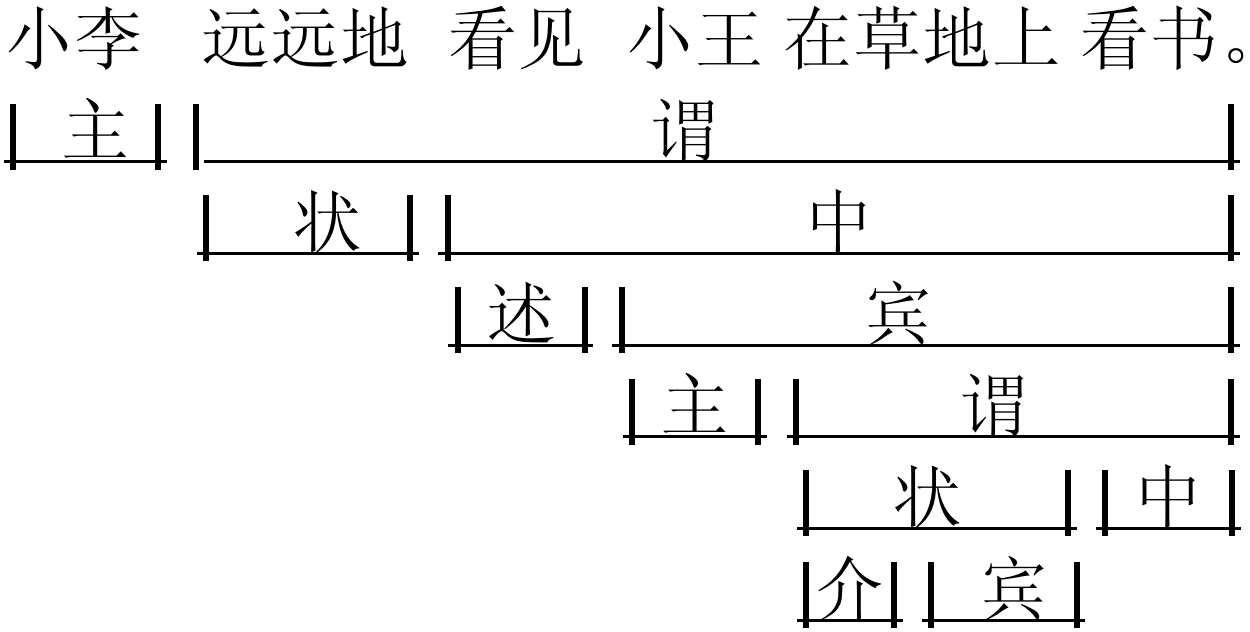
内容提要

句法结构

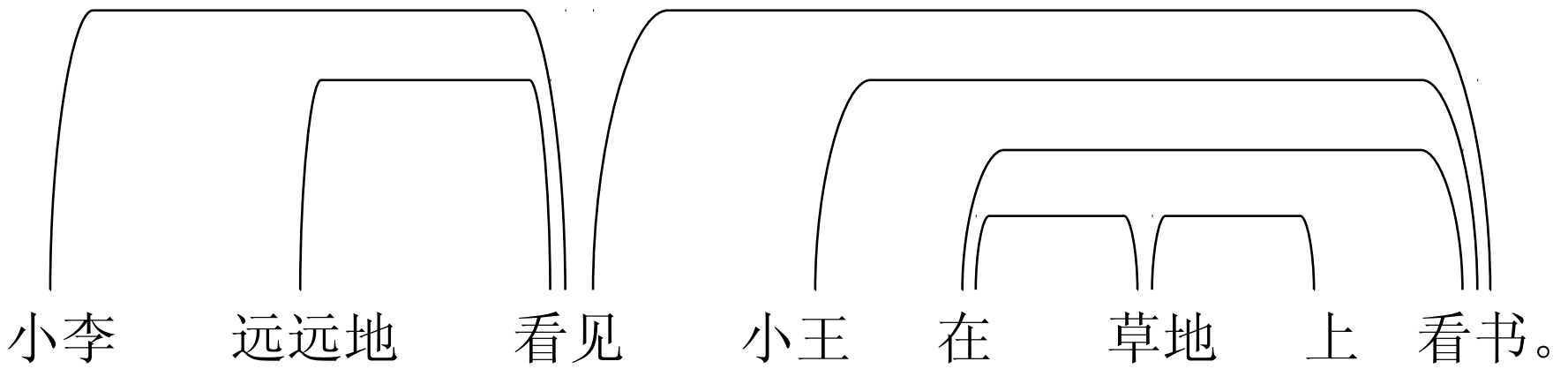
短语结构分析

依存分析

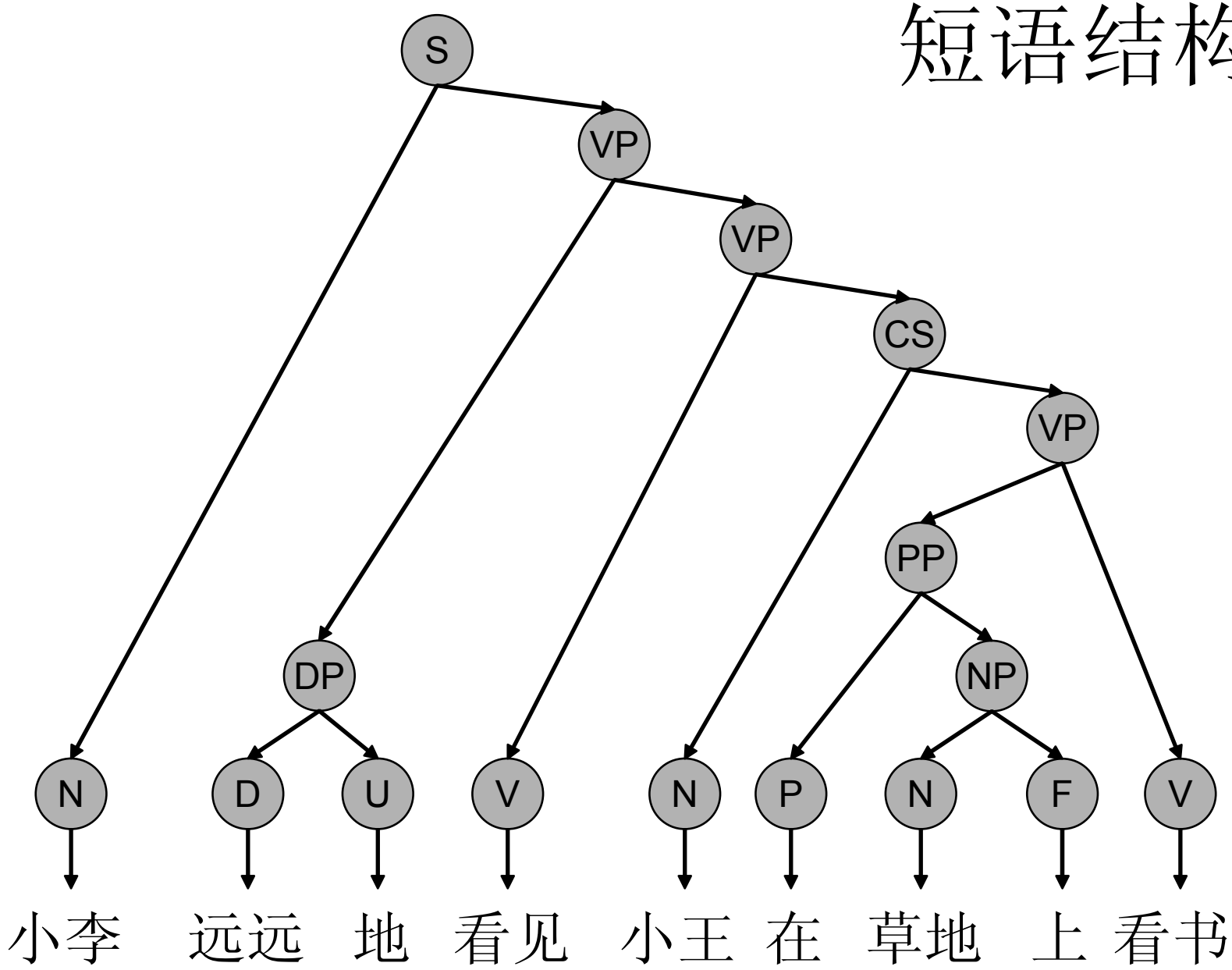
词语到句子的组合顺序



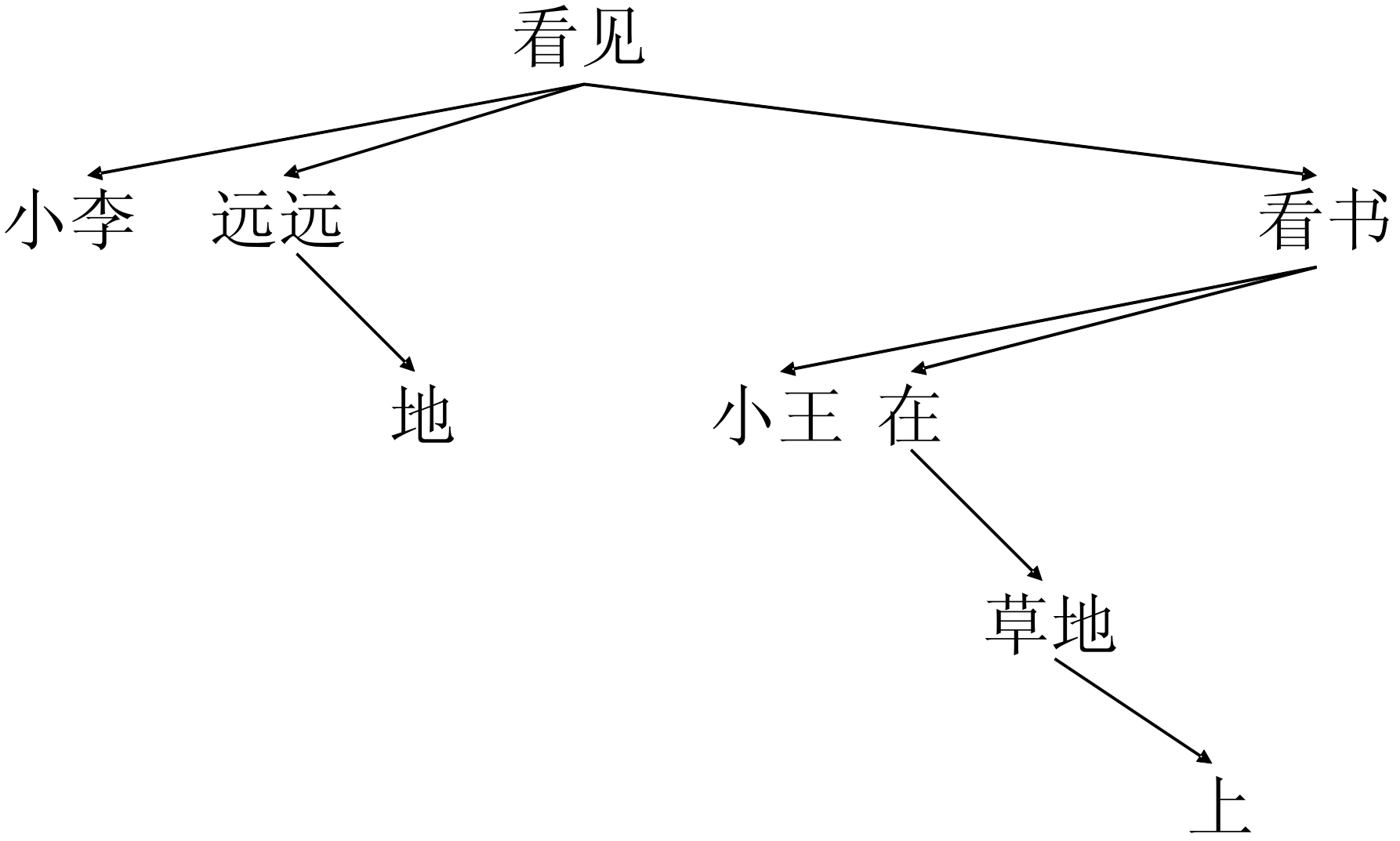
词语之间的依赖关系



短语结构树



依存结构树



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相同；
- 本课程介绍上下文无关语法的句法分析（短语结构分析）和依存语法的句法分析（依存分析）

句法结构歧义的消解（1）

- 人们正常交流中所使用的语言，放在特定的环境下看，一般是没有歧义的，否则人们将无法交流（某些特殊情况如幽默或双关语除外）
- 如果不考虑语言所处的环境和语言单位的上下文，将会发现语言的歧义现象无所不在；
- 结论：一般来说，语言单位的歧义现象在引入更大的上下文范围或者语言环境时总是可以被被消解的。句法分析的核心任务就是消解一个句子在句法结构上的歧义。

句法结构的歧义消解（2）

- 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

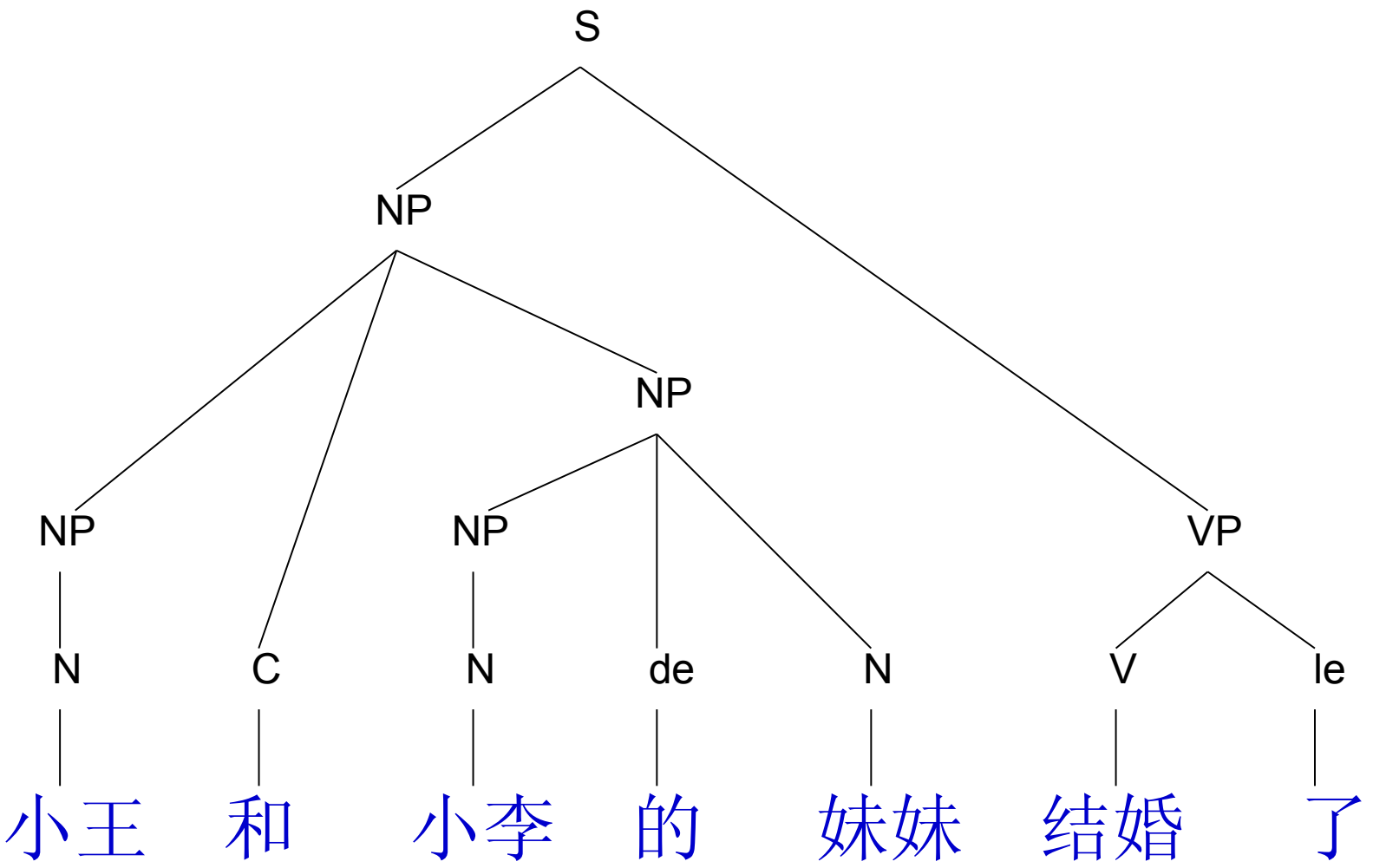
妹妹： N

结婚： V

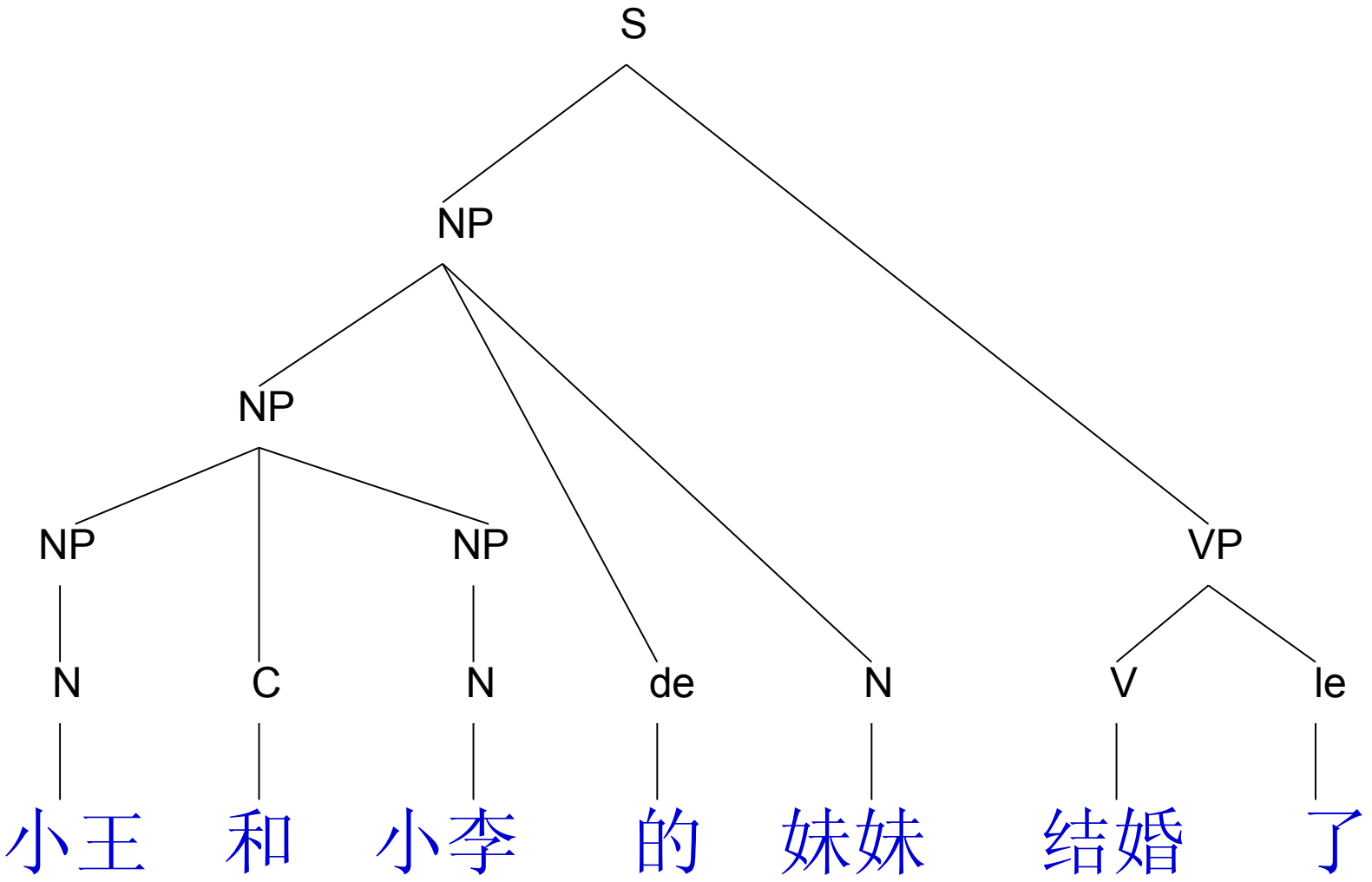
了： le

的： de

例子一分析结果之一



例子一分析结果之二



内容提要

句法结构

短语结构分析

依存分析

短语结构分析

- 问题：
  - 给定一部上下文无关语法
  - 输入一个句子
  - 输出该句子的短语结构树
- 由于短语结构语法（特别是上下文无关语法）应用得最为广泛，因此以短语结构树为目标的句法分析器研究得最为彻底；
- 很多其他形式语法对应的句法分析器都可以通过对短语结构语法的句法分析器进行简单的改造得到。

短语结构分析

移进归约算法

CYK 算法

概率上下文无关语法

词汇化概率上下文无关语法

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

上下文无关语法

规则：

S → NP VP

NP → R

NP → N

NP → Sϕ de

VP → V NP

Sϕ → NP VPϕ

VPϕ → V V

词典：

我： R

县长： N

是： V

派： V

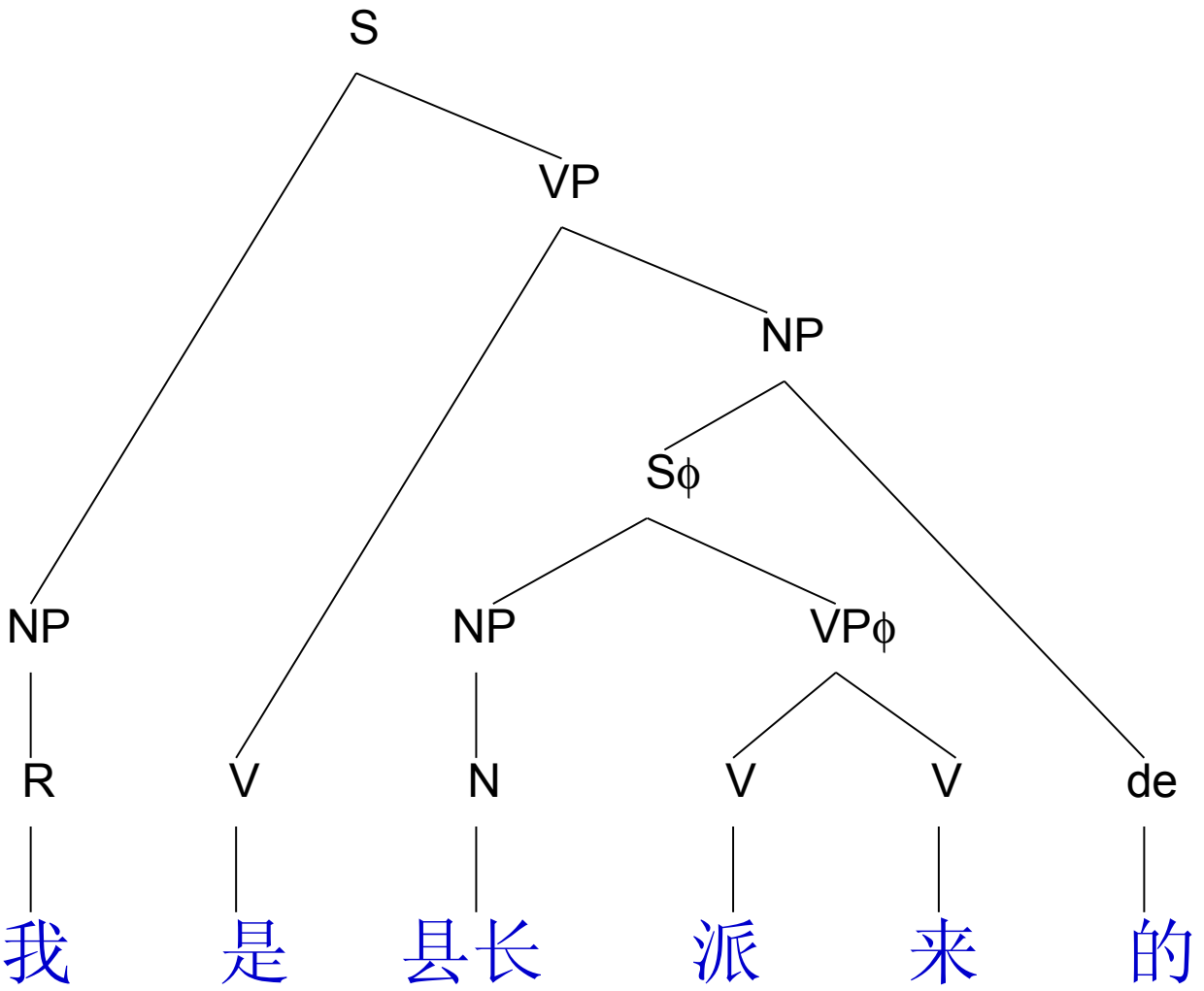
来： V

的： de

短语结构分析——输入

我　　是　　县长　　派　　来　　的

短语结构分析——输出



移进—归约算法：举例

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

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—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
- 可以看到，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—归约冲突和归约—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

短语结构分析

移进归约算法

CYK 算法

概率上下文无关语法

词汇化概率上下文无关语法

CYK 算法—概述

- 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—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。

短语结构分析

移进归约算法

CYK 算法

概率上下文无关语法

词汇化概率上下文无关语法

概率在句法分析中的作用

- 决定最有可能的句子，起到语言模型的作用
- 加速分析器
- 排歧

条件概率建模与联合概率建模

- 用条件概率建模

$$P(t \mid s, G), where \sum_{t} P(t \mid s, G) = 1$$

$$\hat{t} = \arg\max_{t} P(t \mid s, G)$$

- 用联合概率建模

$$\sum_{\{t: yield(t) \in L\}} P(t) = 1 \qquad P(s) = \sum_{t} P(s,t) = \sum_{\{t: yield(t) = s\}} P(t)$$

$$\hat{t} = \arg\max_{t} P(t \mid s) = \arg\max_{t} \frac{P(t,s)}{P(s)} = \arg\max_{t} P(t,s)$$

最简单的概率语法模型

- PCFG：Probabilistic CFG 概率上下文无关语法
  SCFG：Stochastic CFG 随机上下文无关语法
- CFG 的简单概率拓广

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

- 基本假设
  - 位置无关（Place invariance）
  - 上下文无关（Context-free）
  - 祖先无关（Ancestor-free）
- 分析树的概率等于所有施用规则概率之积

例子

$$P\left(\begin{array}{ccc} & {}^1S & \\ & \wedge & \\ {}^2NP & & {}^3VP \\ \wedge & & | \\ the\ man & & snores \end{array}\right)$$

$$= P({}^1S_{13} \rightarrow {}^2NP_{12}\,{}^3VP_{33},\ {}^2NP_{12} \rightarrow the_1 man_2,\ {}^3VP_{33} \rightarrow snores_3)$$

$$= P({}^1S_{13} \rightarrow {}^2NP_{12}\,{}^3VP_{33})P({}^2NP_{12} \rightarrow the_1 man_2 | {}^1S_{13} \rightarrow {}^2NP_{12}\,{}^3VP_{33})$$

$$P({}^3VP_{33} \rightarrow snores_3 | {}^1S_{13} \rightarrow {}^2NP_{12}\,{}^3VP_{33},\ {}^2NP_{12} \rightarrow the_1 man_2)$$

$$= P({}^1S_{13} \rightarrow {}^2NP_{12}\,{}^3VP_{33})P({}^2NP_{12} \rightarrow the_1 man_2)P({}^3VP_{33} \rightarrow snores_3)$$

$$= P(S \rightarrow NP\,VP)P(NP \rightarrow the\,man)P(VP \rightarrow snores)$$

概率上下文无关语法——示例

CFG

S→NP VP
VP → V NP
NP → N
NP → NP 的 NP
NP → VP 的 NP

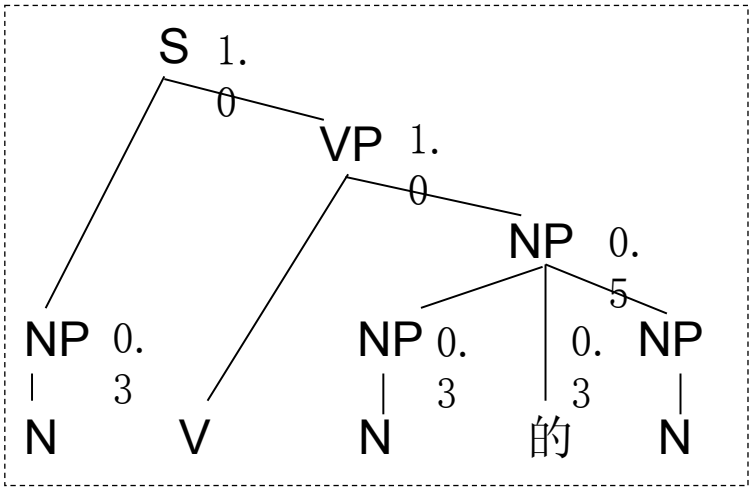
PCFG

S→NP VP	1.0
VP → V NP	1.0
NP → N	0.3
NP → NP 的 NP	0.5
NP → VP 的 NP	0.2

分析树及其概率

老虎 咬死了 猎人 的 狗

N V N 的 N

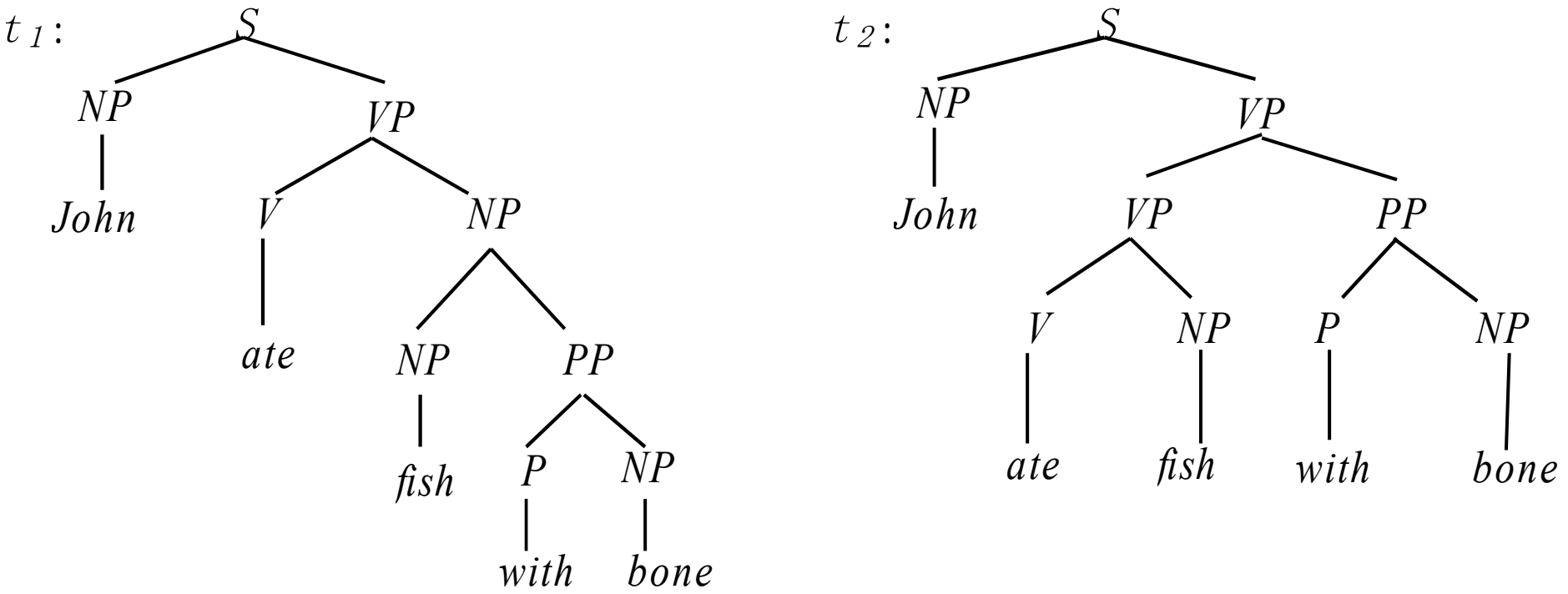


P(S)=1.0×0.3×1.0×0.3×0.5×0.3

=0.0135

用概率来帮助判别歧义

sentence = "John ate fish with bone"



分析树的概率与句子的概率

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow John$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow bone$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow star$	0.04
$P \rightarrow with$	1.0	$NP \rightarrow fish$	0.18
$V \rightarrow ate$	1.0	$NP \rightarrow telescope$	0.1

$$P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18$$
$$= 0.0009072$$

$$P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18$$
$$= 0.0006804$$

$$P(sentence) = P(t_1) + P(t_2) = 0.0015876$$

PCFG 的三个基本问题

- 一个语句 $W=w_1w_2....w_n$ 的 $P(W|G)$, 也就是产生语句 W 的概率？

  评估问题　　向内算法

- 在语句 W 是歧义的情况下, 如何快速选择最佳的语法分析(parse)？

  解码问题　　韦特比算法

- 如何调节 G 的概率参数, 使得 $P(W|G)$ 最大?

  学习问题　　向内向外算法

韦特比算法

- 韦特比变量 $\gamma_{ij}(A)$ 为非终结符 A 经由某一推导而产生 $w_i w_{i+1} \ldots w_j$ 的最大概率， $\Psi(A)$ 为最佳推导。
  - 动态规划公式

$$\gamma_{ii}(A) = \max P(A \Rightarrow w_i)$$

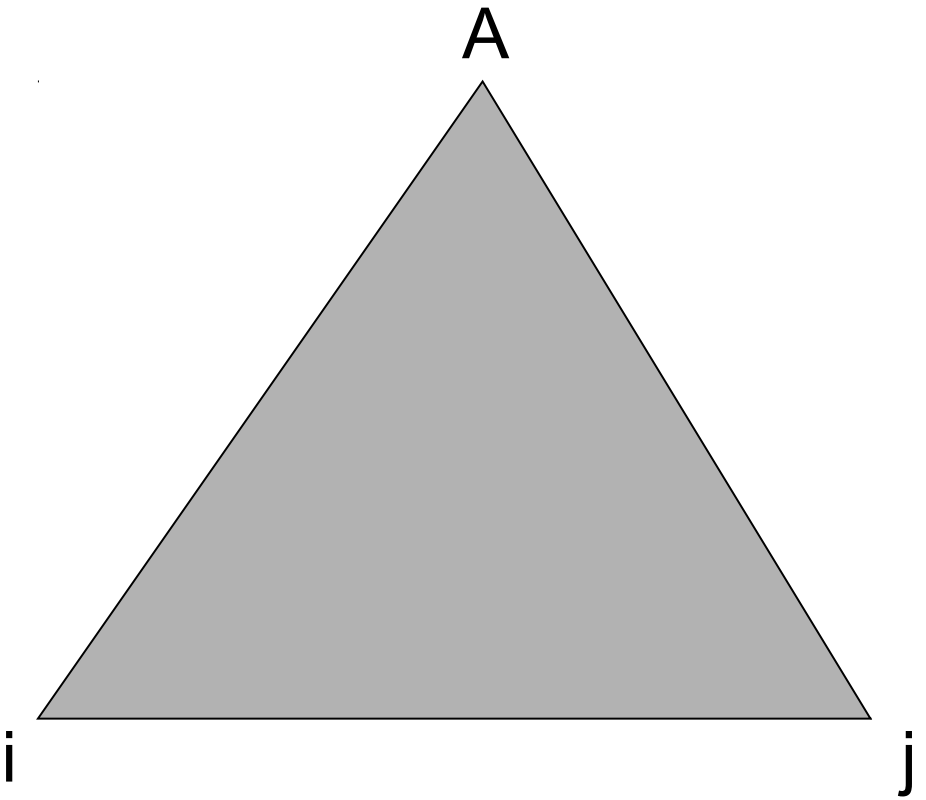
$$\gamma_{ij}(A) = \max_{B,C \in N; i \le k \le j} P(A \rightarrow BC)\gamma_{ik}(B)\gamma_{(k+1)j}(C)$$

$$\psi_{ij}(A) = \arg\max_{B,C \in N; i \le k \le j} P(A \rightarrow BC)\gamma_{ik}(B)\gamma_{(k+1)j}(C)$$

  - 特别地，

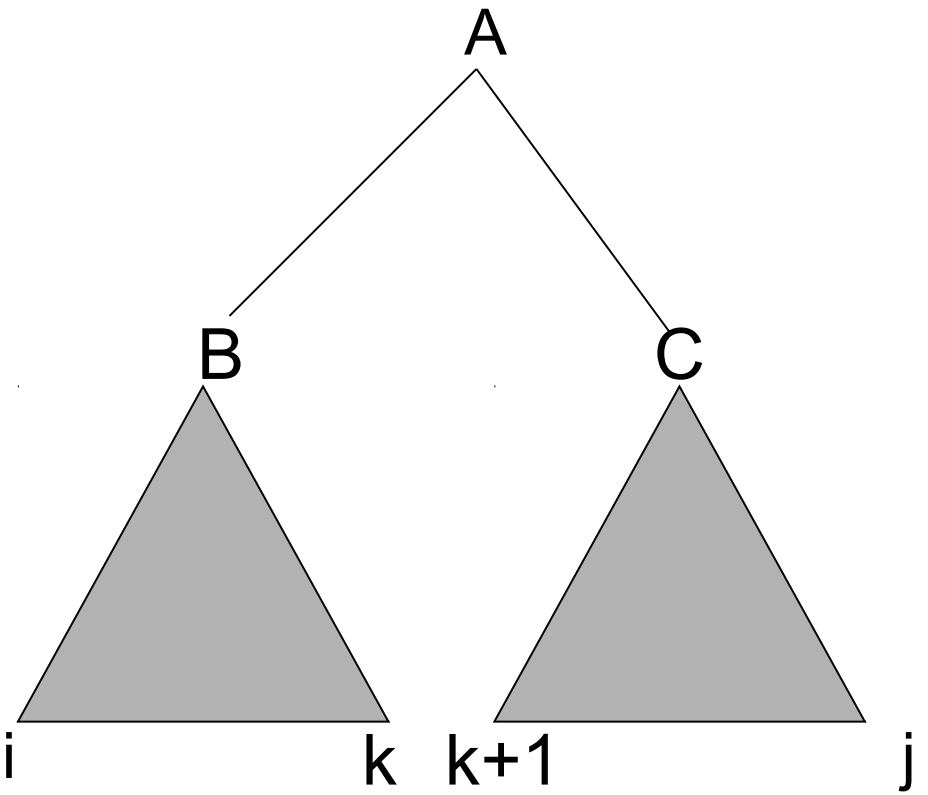
$$\gamma_{1n}(S)$$

韦特比变量



$$\gamma_{ii}(A) = \max P(A \Rightarrow w_i)$$

韦特比变量的递归计算



$$\gamma_{ij}(A) = \max_{B,C \in N; i \le k \le j} P(A \to BC)\gamma_{ik}(B)\gamma_{(k+1)j}(C)$$

PCFG 的优点

- 化解结构歧义(structurally different parses)
- 加速语法分析(尽早删除小概率子结构)
- 增强分析器鲁棒性(use of low probabilities)
- 定量比较语法(language model)
- 便于语法归纳(grammar induction)

PCFG 的缺点

- 合理性差(单纯依据结构给出概率估计)
- 不如 n 元语法(importance of lexical context)
- 明显的偏向性(smaller tree, small number of expansions will be favored)

短语结构分析

移进归约算法

CYK 算法

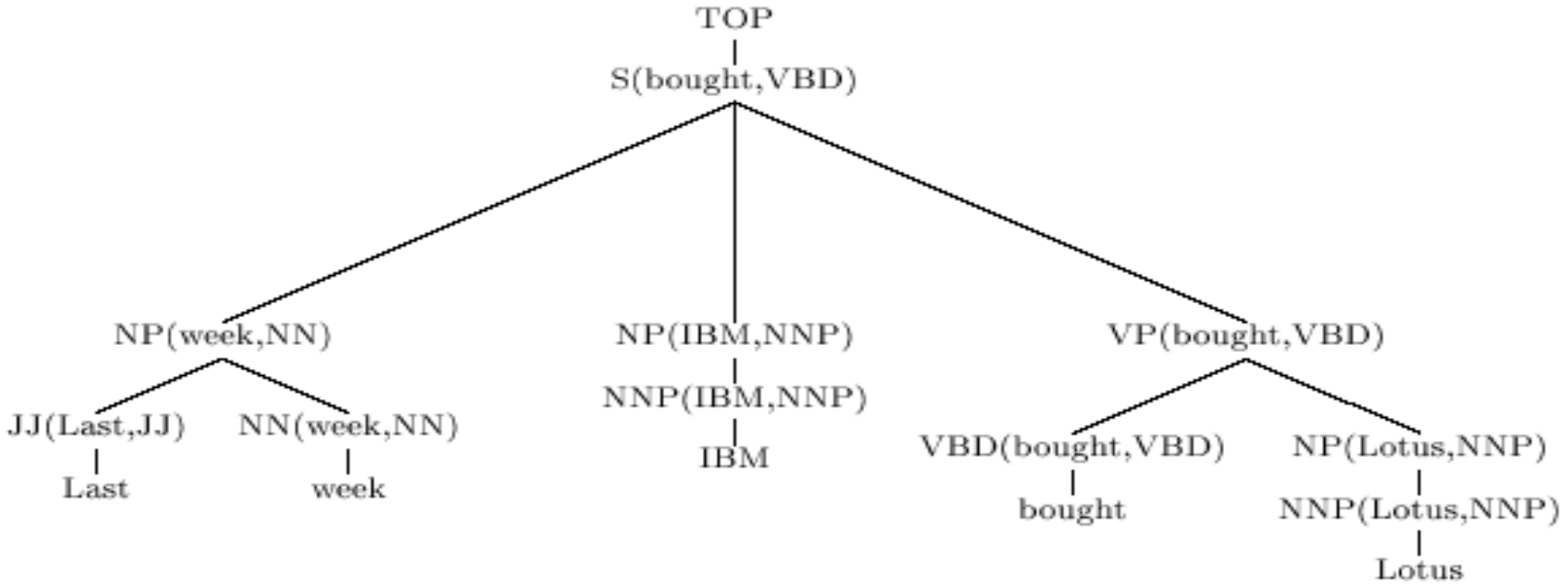
概率上下文无关语法

词汇化概率上下文无关语法

Lexicalized PCFG

- 词汇化上下文概率上下文无关语法 Lexicalized PCFG
- 每一个非终结符被关联到一个中心词 w 和一个中心词形 t

Lexicalized PCFG



Internal Rules:

```
TOP                 →   S(bought,VBD)
S(bought,VBD)       →   NP(week,NN)      NP(IBM,NNP)      VP(bought,VBD)
NP(week,NN)         →   JJ(Last,JJ)      NN(week,NN)
NP(IBM,NNP)         →   NNP(IBM,NNP)
VP(bought,VBD)      →   VBD(bought,VBD)  NP(Lotus,NNP)
NP(Lotus,NNP)       →   NNP(Lotus,NNP)
```

Lexical Rules:

```
JJ(Last,JJ)         →   Last
NN(week,NN)         →   week
NNP(IBM,NNP)        →   IBM
VBD(bought,VBD)     →   bought
NNP(Lotus,NN)       →   Lotus
```

Collins Model (1)

- Michael Collins. Head-Driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania, 1999.
- 复杂程度递增的三个模型
  - Model 1 : lexical dependency
  - Model 2 : complement/adjunct distinction, subcat frame
  - Model 3: Wh-movement

Collins Model (2) 中心成分的生成

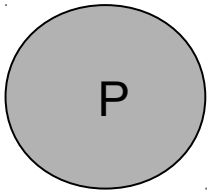
- 两个组成部分：词汇中心和结构中心
- 词汇中心：中心词(hw)和中心词词性标记(ht)
- 结构中心：中心成分的短语标记(hn)
- 中心成分的生成
  - 首先生成词汇中心
  - 其次生成结构中心

Collins Model (3) 修饰成分的扩展

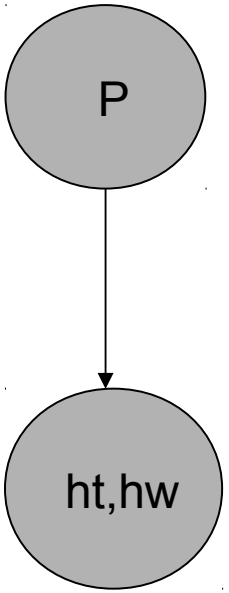
- 基本规则形式
  $P(h) \rightarrow L_m(l_m)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_n(r_n)$
- 修饰成分 $M_i(m_i)$ 扩展
  - 修饰成分词汇中心的扩展：根据 P(h) 和 H(h) 估计 M_i 的词汇中心
  - 结构扩展：估计 M_i 的标记

$M_i(m_i)$ 为 $L_i(l_i)$ 和 $R_i(r_i)$ 的统称

Collins Model (3a) 图示

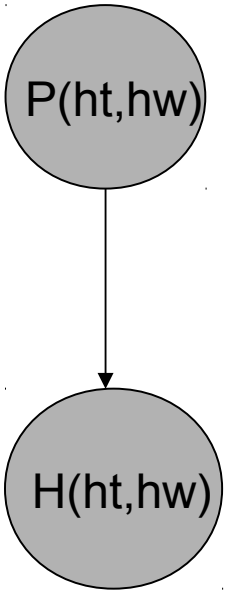


Collins Model (3b) 图示



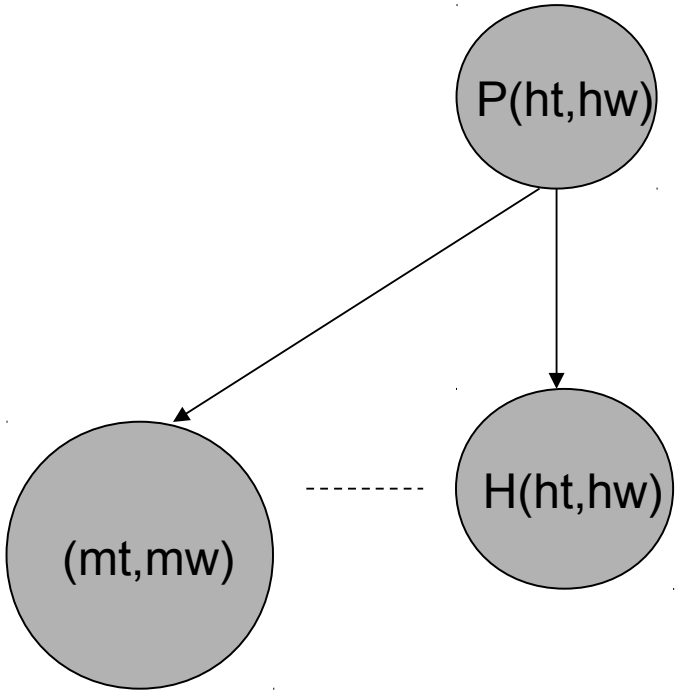
1. generate (ht,hw)

Collins Model (3c) 图示



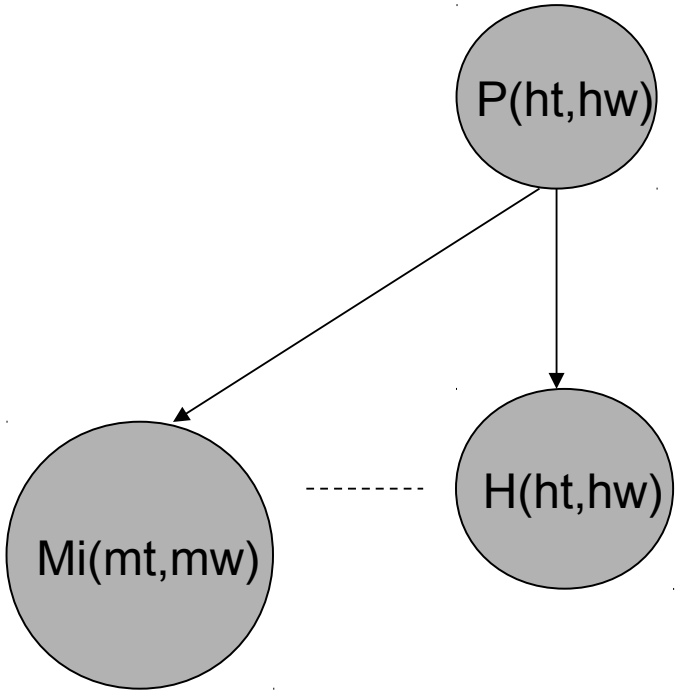
1. generate (ht,hw)
2. generate H

Collins Model (3d) 图示



1. generate (ht,hw)
2. generate H
3. expand (mt,mw)

Collins Model (3e) 图示



1. generate (ht,hw)
2. generate H
3. expand (mt,mw)
4. expand Mi

Collins Model (4) 模型的集成

- 根据当前短语标记生成中心词汇信息：
  p(ht,hw|P)
- 根据结构信息和词汇信息生成中心短语标记：
  p(H|P,ht,hw).
- 根据词汇依赖信息和结构信息生成修饰成分的词汇信息：
  p(mt,mw|P,H,ht,hw,dist,dir)
- 根据词汇信息和结构依赖信息生成修饰成分的短语标记：
  p(Mi|mt,mw,ht,hw,P,H,dis,dir)

上面 dist 是修饰成分到中心成分的距离， dir 是修饰成分对于中心成分的方向（左或右）

Collins Model (5) 模型的集成

$$p(P(h) \rightarrow L_m(l_m)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_n(r_n))$$

$$= p(ht,hw|P) * p(H|P,ht,hw)$$

$$* \Pi_{dist,dir}(p(mt_{dist,dir},mw_{dist,dir}|P,H,ht,hw,dist,dir)$$

$$* p(M_{dist,dir}|mt,mw,ht,hw,P,H,dis,dir))$$

其中： $h=(ht,hw)$

$$L_{dist}(l_{dist})=M_{dist,left}(mt_{dist,left},mw_{dist,left})$$

$$R_{dist}(r_{dist})=M_{dist,right}(mt_{dist,right},mw_{dist,right})$$

Collins Model (6) 参数估计

- Collins Model 数据稀疏问题极为严重
- 采用回退法进行参数平滑
  - 对概率的条件部分逐级回退
  - 最坏情况下回退到基本的 PCFG
- 可以看出， Collins Model 是典型的生成模型

Collins Model (7) 中心成分确定

- 中心成分映射规则示例
  - 规则： IP right { IP VP }
  - 意义：对于句法树中标识为 IP 的节点，自右向左扫描该节点的所有孩子，第一个出现在列表{ IP VP } 中的孩子即为中心孩子节点。父节点的中心词和中心词词性等价于其中心孩子结点的中心词和中心词词性

概率句法分析的训练与评价

- 采用 Treebank 作为训练和测试语料库
- 目前研究界普遍采用 Penn Treebank（宾州树库）作为训练和测试语料库（汉语和英语），并且有公认的语料库划分标准
- 句法分析的评价通常采用标记正确率（label precision）和标记召回率（label recall）
- 在宾州树库上，英语句法分析的标记正确率和标记召回率比汉语句法分析的标记正确率和标记召回率大约高出5-10%

内容提要

句法结构

短语结构分析

依存分析

依存分析

依存分析介绍

最大生成树模型

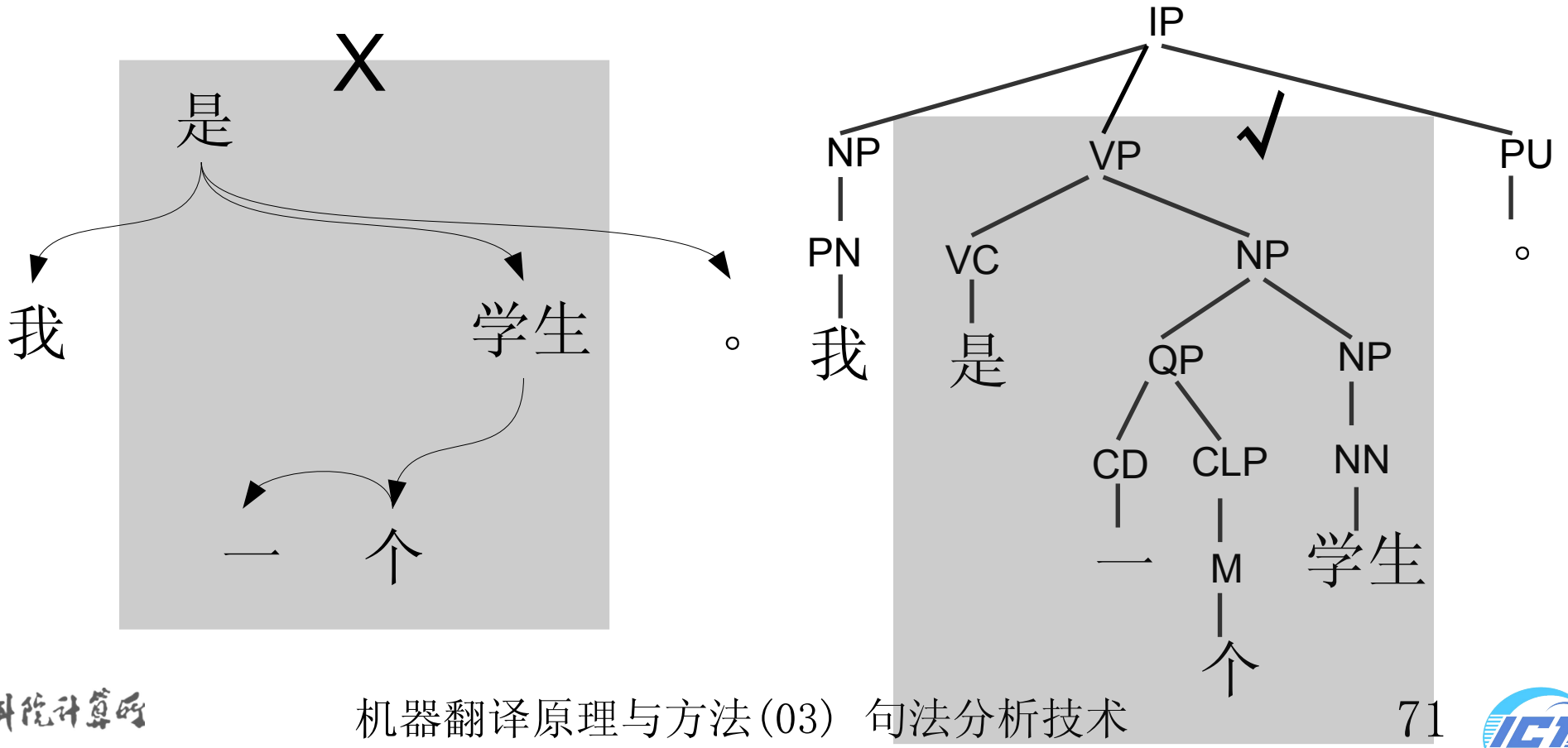
移进归约状态转移模型

依存分析

- 依存结构和依存语法
- 短语结构树转依存树
- 依存分析模型
  - 概率依存模型
  - 最大生成树模型
  - 状态转移模型

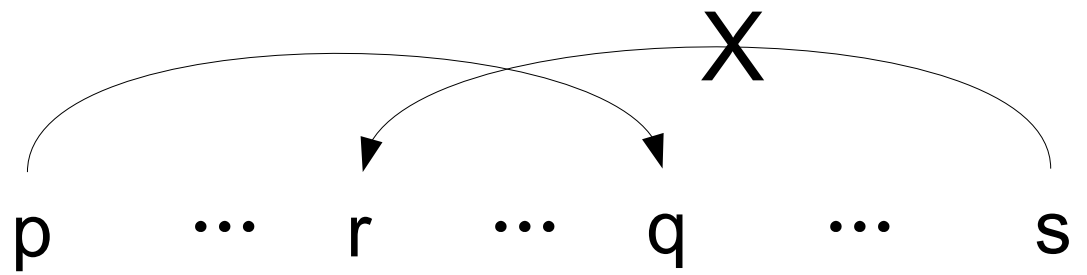
依存分析简介

- 依存分析与短语结构分析类似，但有所不同：依存分析丢掉了跨度信息和跨度上的句法标识



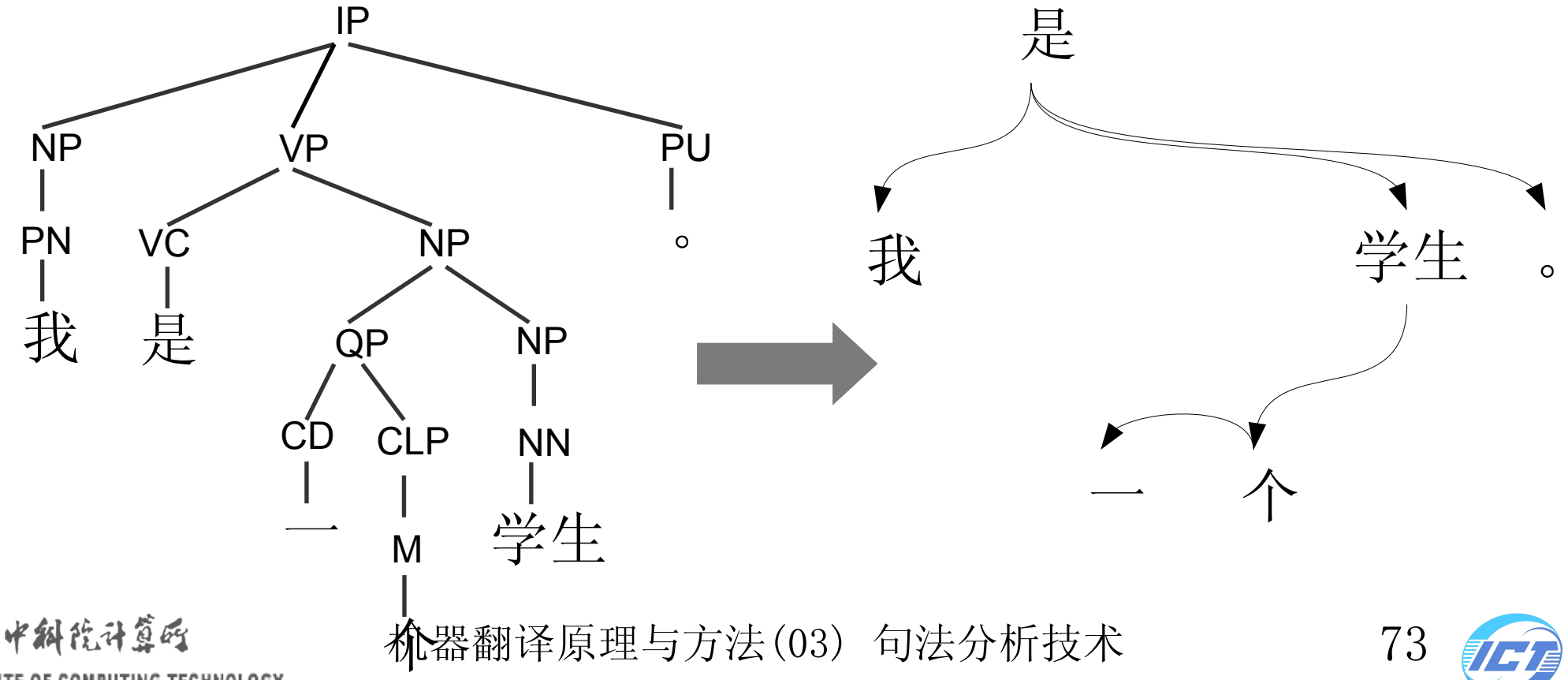
依存分析简介

- 大多数语言，包括汉语和英语，满足投射性。所谓投射性是指：如果词 p 依存于词 q，那么 p 和 q 之间的任意词 r 就不能依存到 p 和 q 所构成的跨度之外



短语结构树转依存树

- 任何短语结构树句法分析模型输出的句法树，通过Yamada and Matsumoto (2003) 的中心词映射规则即可转化为依存结构树



短语结构树转依存树

- 中心词映射规则示例
  - 规则： IP right { IP VP }
  - 意义：对于句法树中标识为 IP 的节点，自右向左扫描该节点的所有孩子，第一个出现在列表{ IP VP }中的孩子即为中心孩子节点。其他孩子节点的中心词将依存到中心孩子节点的中心词
- 对于给定的短语结构树，自底向上应用中心词映射规则，即可确定各词之间的依存关系

依存分析

依存分析介绍

最大生成树模型

移进归约状态转移模型

最大生成树模型

- McDonald et al., 2005
- McDonald and Pereira, 2006
- 给定一个包含 N 个词的句子，任意两个词之间都可能存在依存关系，共有 N*(N-1) 种可能的依存边（不能含有依存到自己的自环），只是依存强弱不同
- 将依存强弱表示为这个完全图中边的分数。于是，寻找最可能的依存树的任务就转化为寻找这个完全图的最大生成树

最大生成树模型—解码搜索

- Chu-Liu-Edmonds 算法：最大生成树算法

参数训练

- 采用感知机模型进行参数训练
- 基本思想：
  - 初始化：特征权重初始化
  - 循环：直到收敛
    - 循环：对每个句子
      - 利用现有权重计算最大生成树
      - 对正确边：权重奖励
      - 对错误边：权重惩罚

最大生成树模型

- 每条边 p→c 的分数定义为

$$\text{score}(p \to c) = f(p \to c) \cdot w$$

- f() 函数返回依存边 p→c 的特征向量； w 为权重向量，它由判别式训练得到

最大生成树模型—特征设计

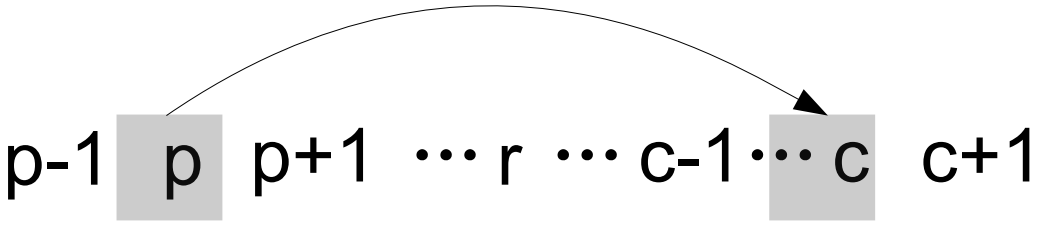
- 除以上特征本身，所有特征都加上父亲-儿子的顺序 ord 和距离 dis， 构成一组新的，更细化的特征
- Ord = (Index(p) > Index(c)) ? Left : Right
- Dis = abs(Index(p) – Index(c))

最大生成树模型—特征设计

- 特征设计针对边进行，而非节点
- 任意一条 p→c 的特征可以取那些呢？

最大生成树模型—特征设计

- 一元特征



Pword, Ppos

Pword

Ppos

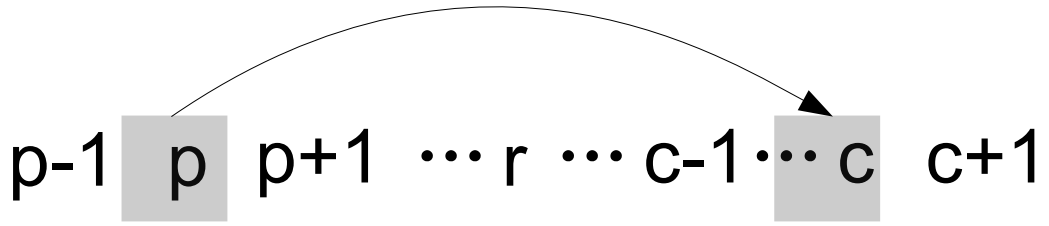
Cword, Cpos

Cword

Cpos

最大生成树模型—特征设计

- 二元特征



Pword, Ppos, Cword, Cpos

Ppos, Cword, Cpos

Pword, Cword, Cpos

Pword, Ppos, Cpos

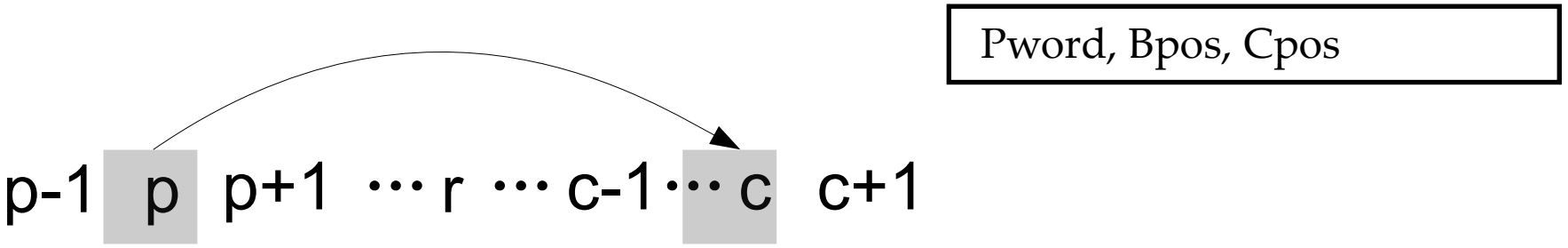
Pword, Ppos, Cword

Pword, Cword

Ppos, Cpos

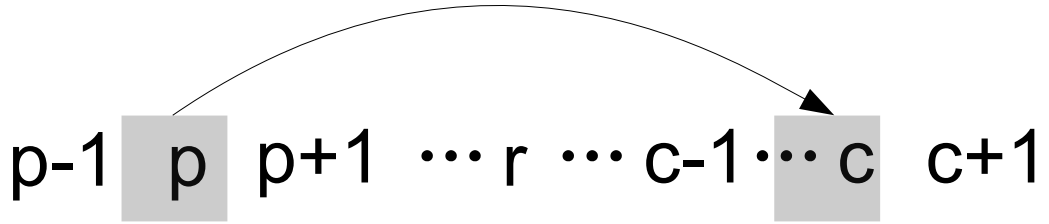
最大生成树模型—特征设计

- 词间词性标注特征， Bpos 为父亲 p 和儿子 c 之间的一个词的词性标注



最大生成树模型—特征设计

- 父亲儿子周围词性标注特征



Ppos,Ppos+1,Cpos-1, Cpos

Ppos-1, Ppos, Cpos-1, Cpos

Ppos, Ppos+1, Cpos, Cpos+1

依存分析

依存分析介绍

最大生成树模型

移进归约状态转移模型

移进规约状态转移模型

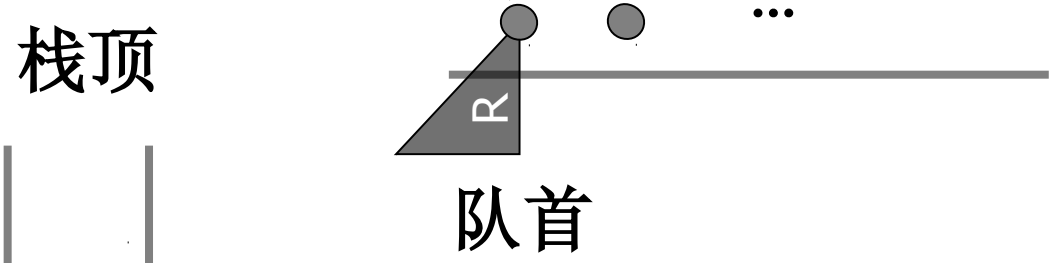
- Nivre et al. , 2006
- 基本思想
- 状态和转移
- 训练和解码
- nbest 全局训练

移进归约状态转移模型—基本思想

- 在分析过程中维护一个堆栈和一个队列，堆栈用以存储到目前为止所有的依存子树，队列存储尚未被分析到的词。堆栈顶端和队列的头部确定了当前分析器的状态，依据该状态决定进行移进、规约或者建立栈顶元素与队首元素的依存关系的操作，从而转入新的状态

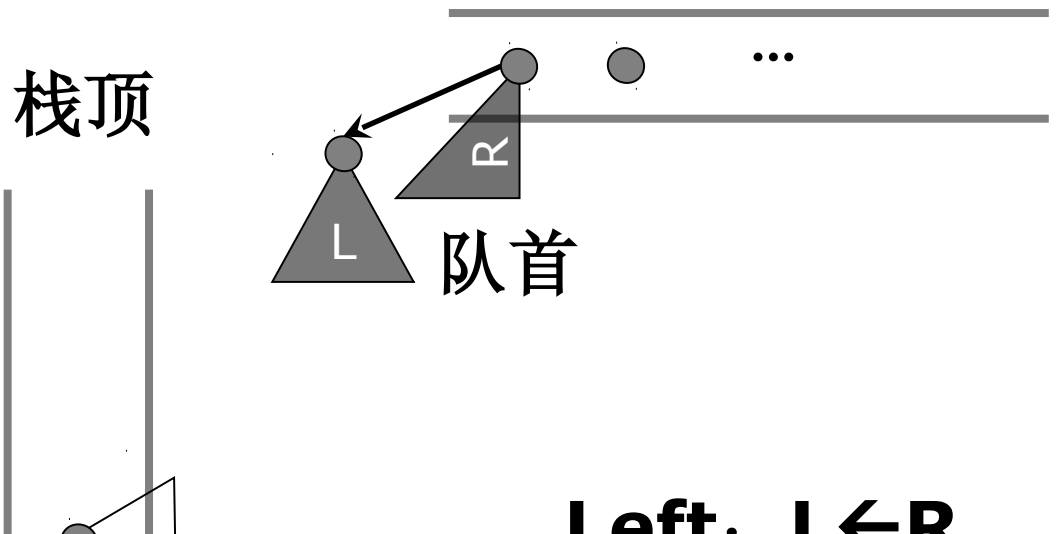
状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



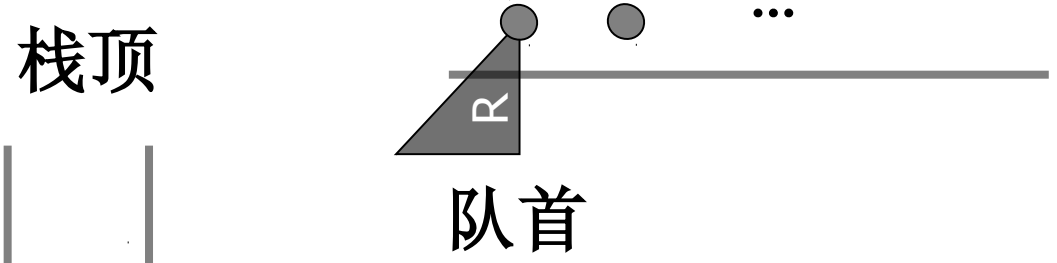
状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



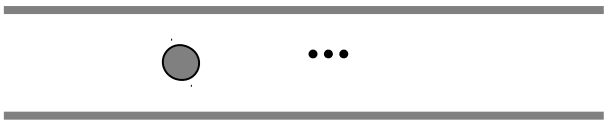
状态和转移

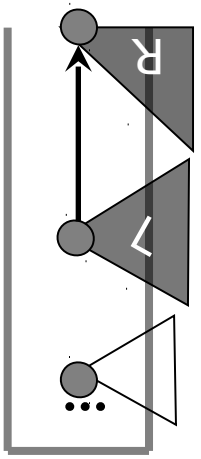
- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



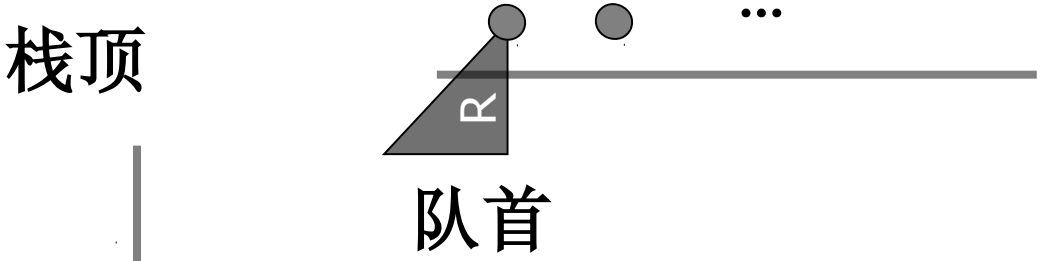


队首

Right：L→R

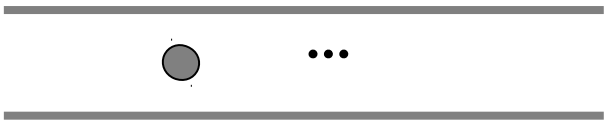
状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce

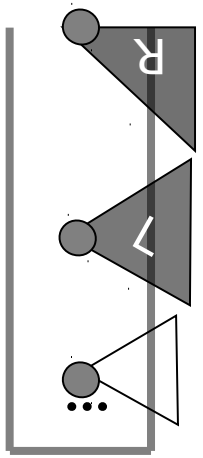


状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



栈顶

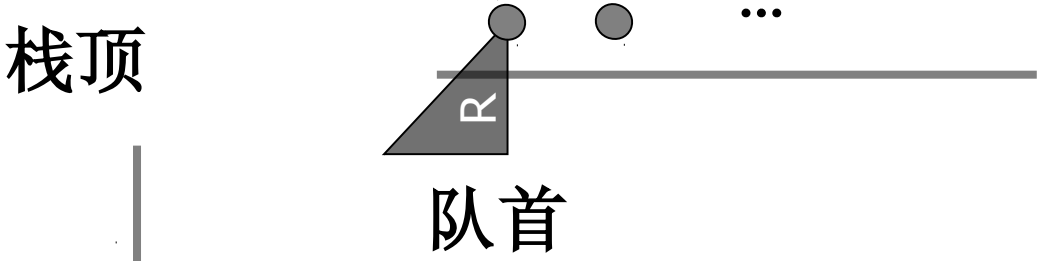


队首

Shift

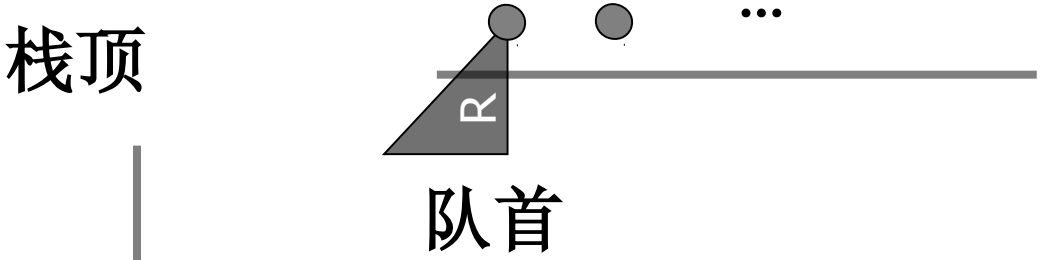
状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



状态和转移

- 在逐趟扫描的过程中，任意时刻的状态是指：当前堆栈和当前队列的格局
- 任意状态下，四种转移动作： left，right，shift 和 reduce



Reduce

状态和转移

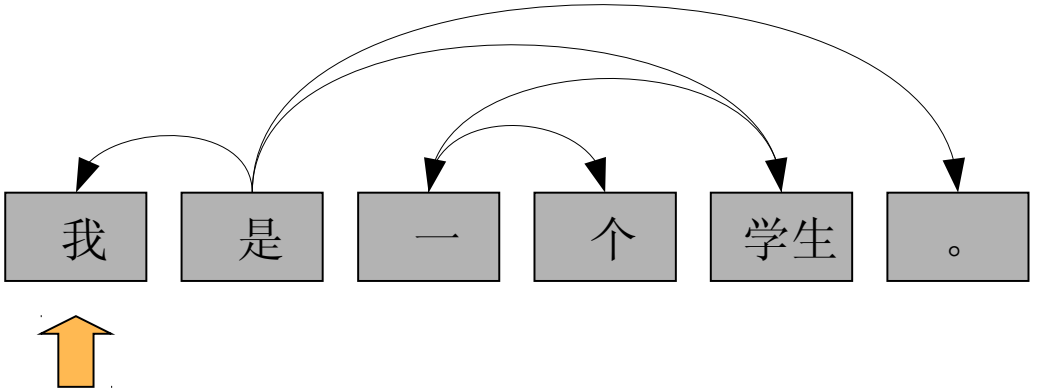
- 动作可以执行的条件
  - Left： 栈顶元素尚无 head
  - Right： 无
  - Shift： 输入队列不空
  - Reduce： 栈顶元素有 head
- 想一想，为什么？

训练和解码

- 训练：
  - 状态转移实例抽取——模拟分析树库中每棵依存树
  - 判别式训练分类器，如 SVM
- 解码：
  在单趟扫描的过程中，分类器根据当前栈顶、当前队首以及栈顶和队首所处的上下文，预测需要执行的状态转移操作并转移到新的状态。这一过程持续至输入队列为空
- 问题：可能最后不能得到完整依存树
- 解决办法？

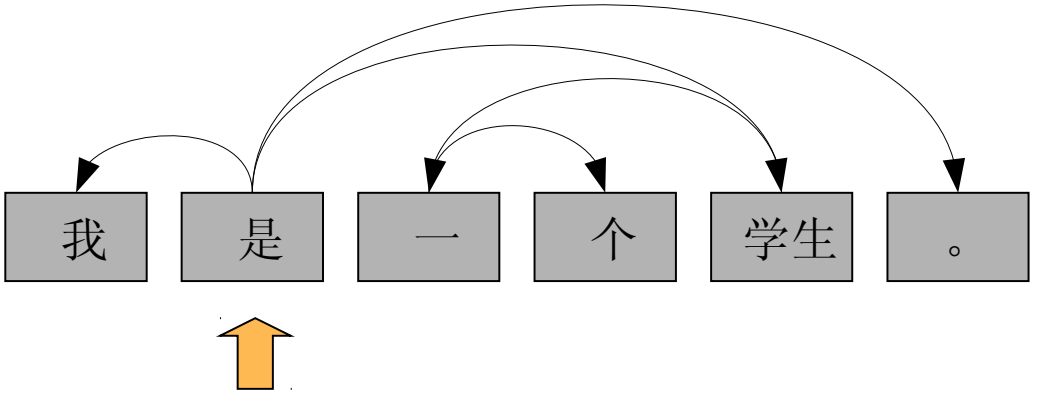
模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M)) 学生/ NN) (。/PU))

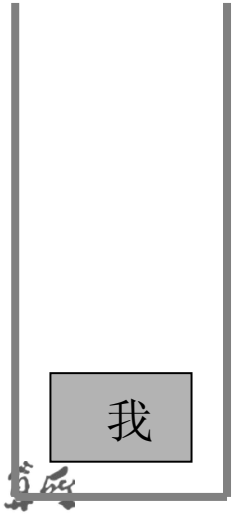


模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M)）学生/ NN)（。/PU))

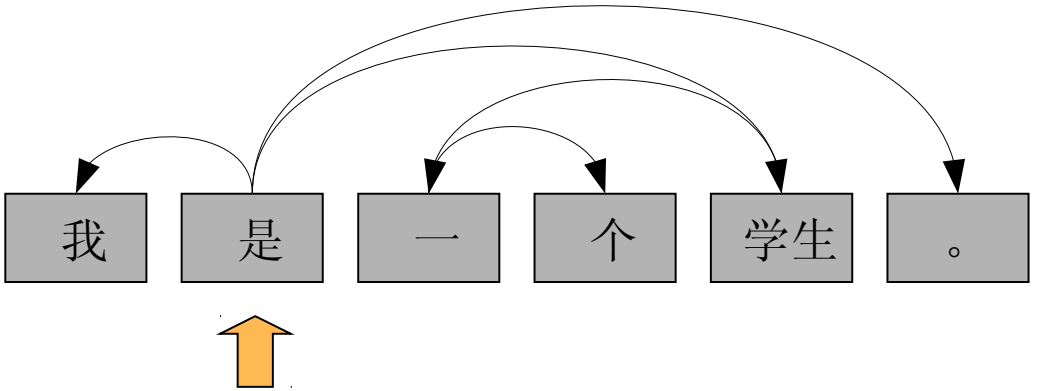


Shift(我)



模拟分析

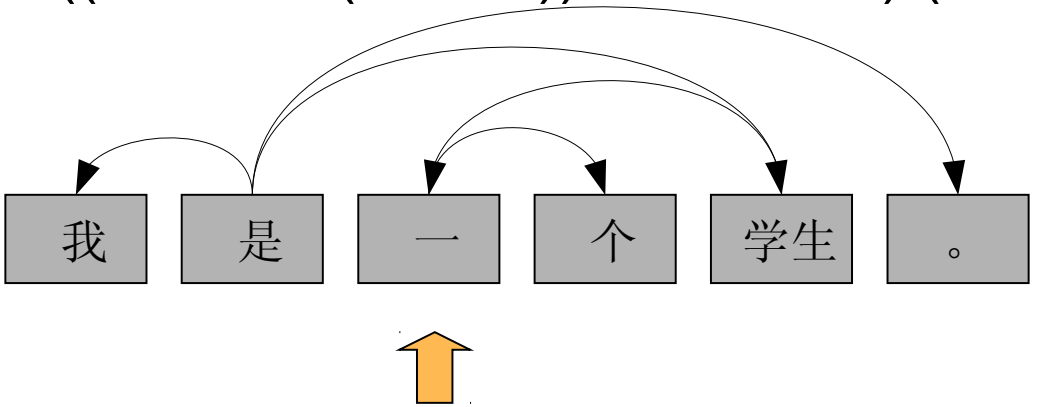
- ((我/ PN) 是/ VC ((一/ CD (个/ M)) 学生/ NN) (。/PU))

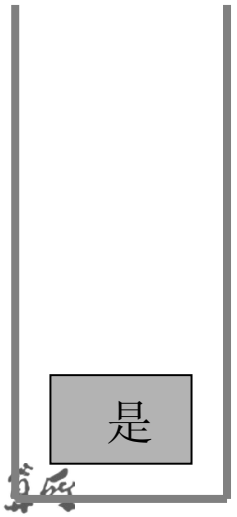


Left(我←是)

模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M)) 学生/ NN)（。/PU))

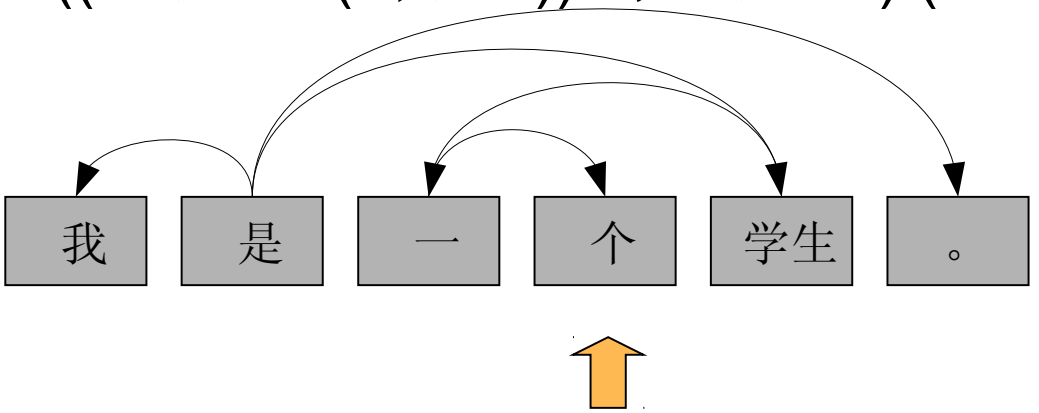


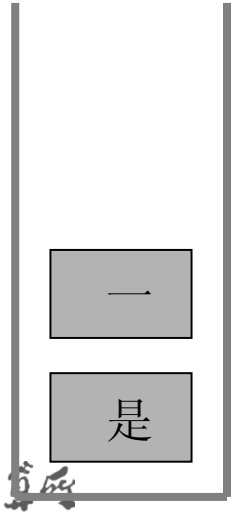


Shift(是)

模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M））学生/ NN）（。/PU））

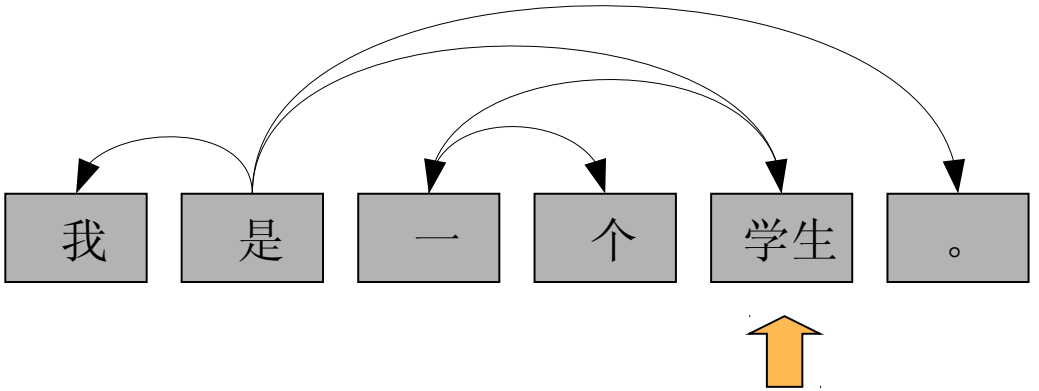


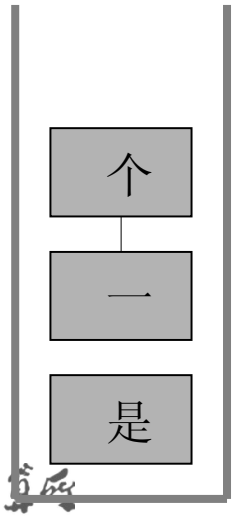


Shift(一)

模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M)) 学生/ NN) (。/PU))

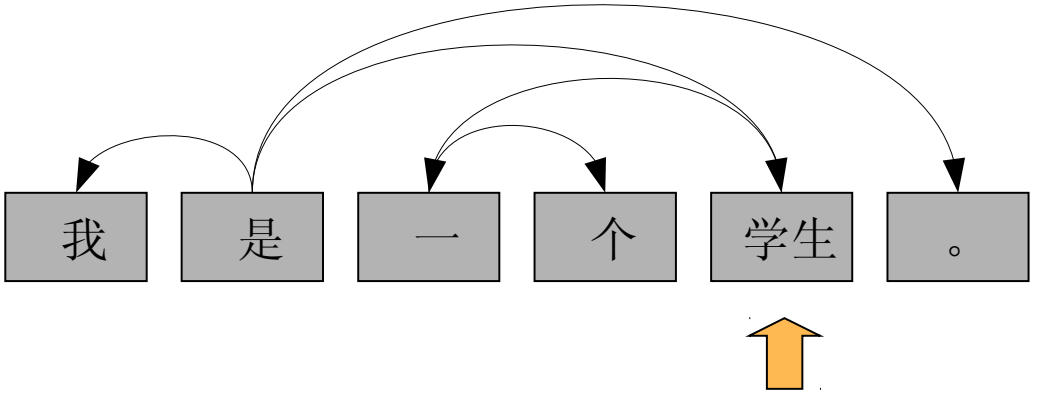


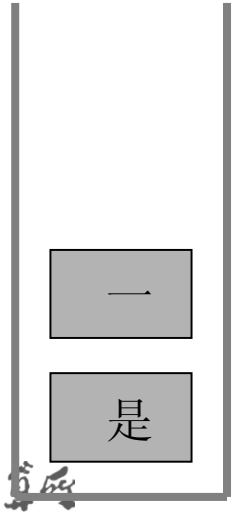


Right(一→个)

模拟分析

- ((我/ PN) 是/ VC ((一/ CD (个/ M)) 学生/ NN) (。/PU))

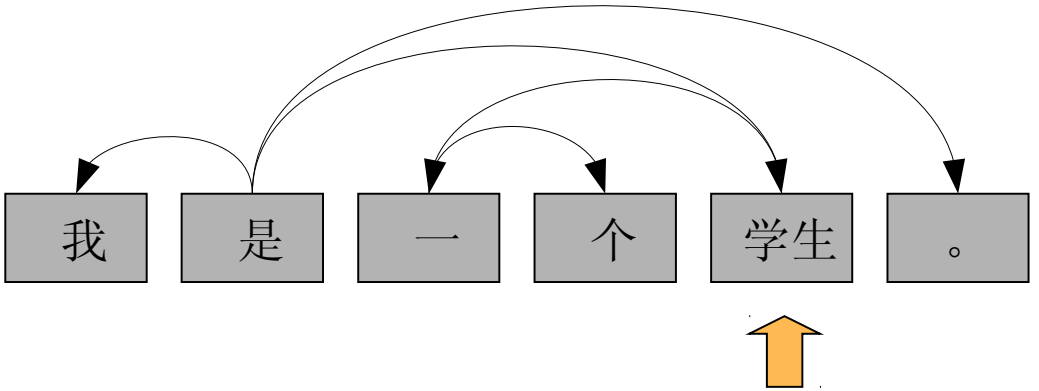


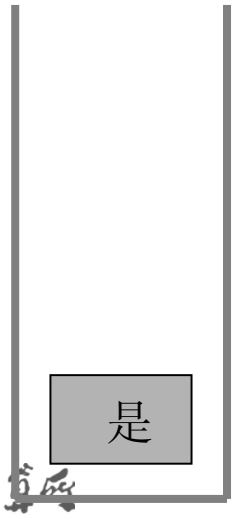


Reduce(个)

模拟分析

- ((我/ PN) 是/ VC ((一/ CD (个/ M)) 学生/ NN) (。/PU))

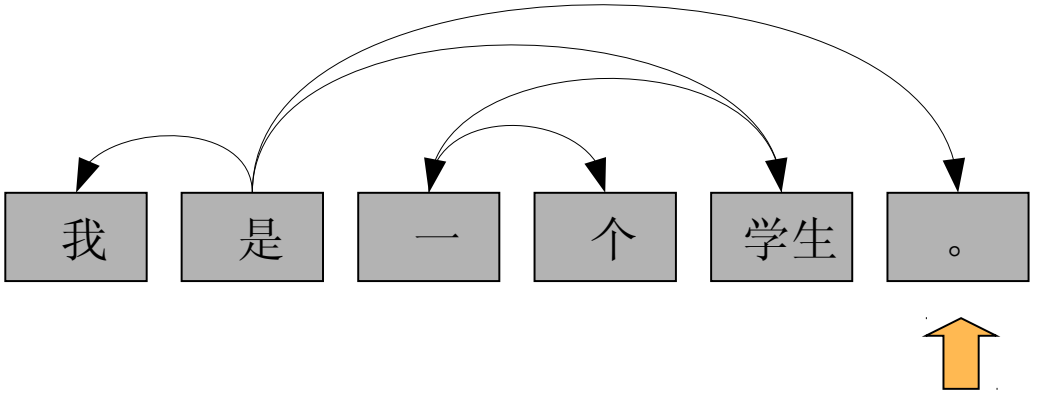


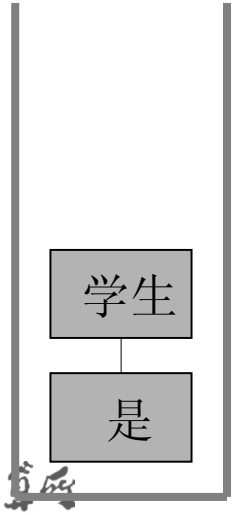


Left(一←学生)

模拟分析

- ((我/ PN) 是/ VC ((一/ CD (个/ M)) 学生/ NN) (。/PU))

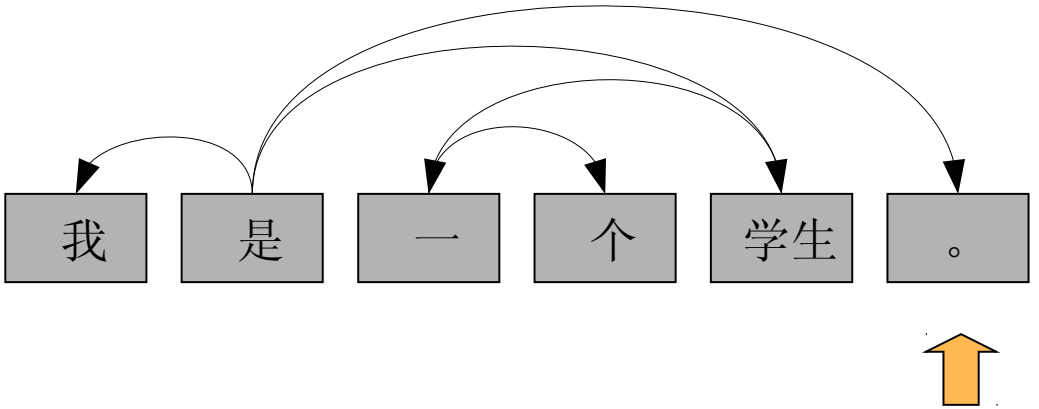


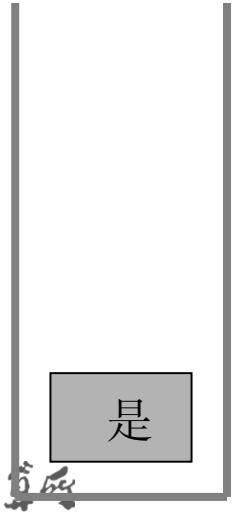


Right(是→学生)

模拟分析

- ((我/ PN) 是/ VC ((一/ CD (个/ M)) 学生/ NN) (。/PU))

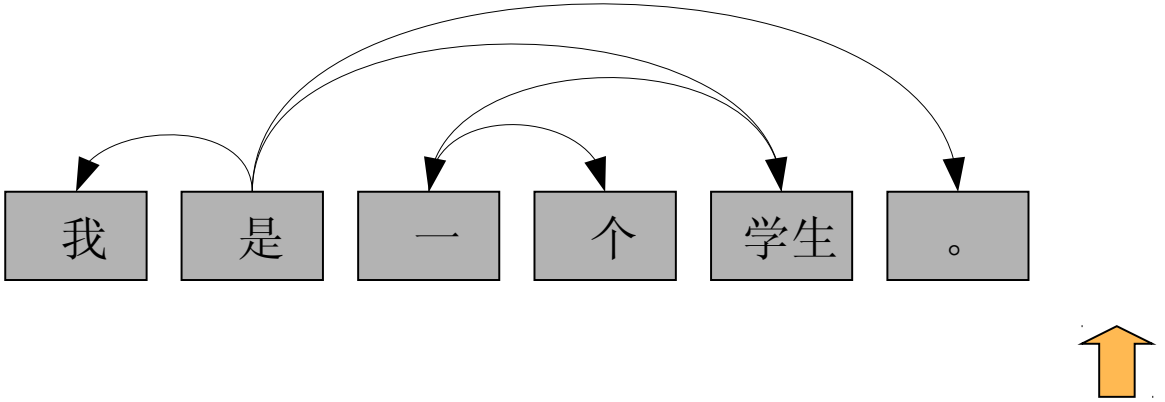


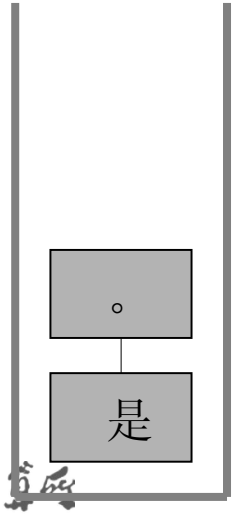


Reduce(学生)

模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M））学生/ NN）（。/PU））

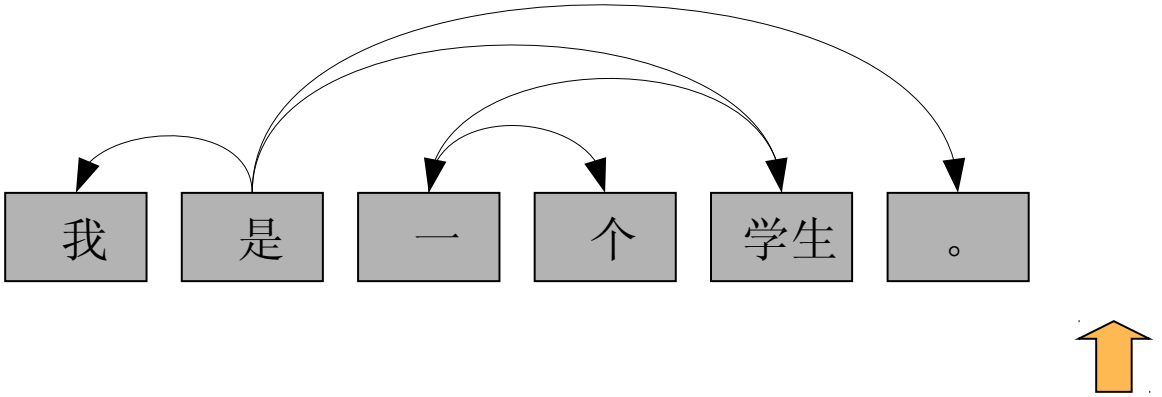


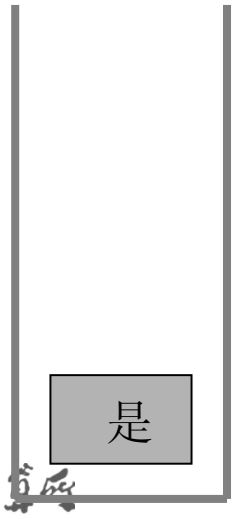


Right(是→。）

模拟分析

- ((我/ PN) 是/ VC ((一/ CD (个/ M)) 学生/ NN) (。/PU))

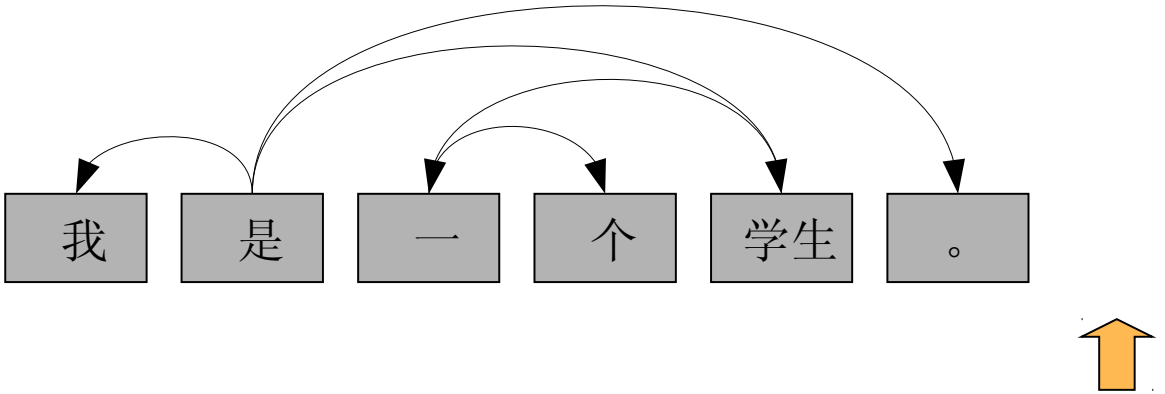




Reduce(。)

模拟分析

- （（我/ PN）是/ VC ((一/ CD（个/ M)) 学生/ NN) (。/PU))



Reduce(是)

nbest 全局训练

- 单 best 局部训练，错误传播严重。试考虑：
  - 每步保留 N 个最好状态
  - 改采用在线全局训练
- nbest 全局训练，使得模型预测分数具有比较归一，便于整合其他模型

（需进一步解释）

依存分析模型比较

- 目前流行的是最大生成树模型和移进规约模型
- 最大生成树模型擅于确定远距离的依存关系
- 移进规约模型对近距离依存关系识别准确率更高