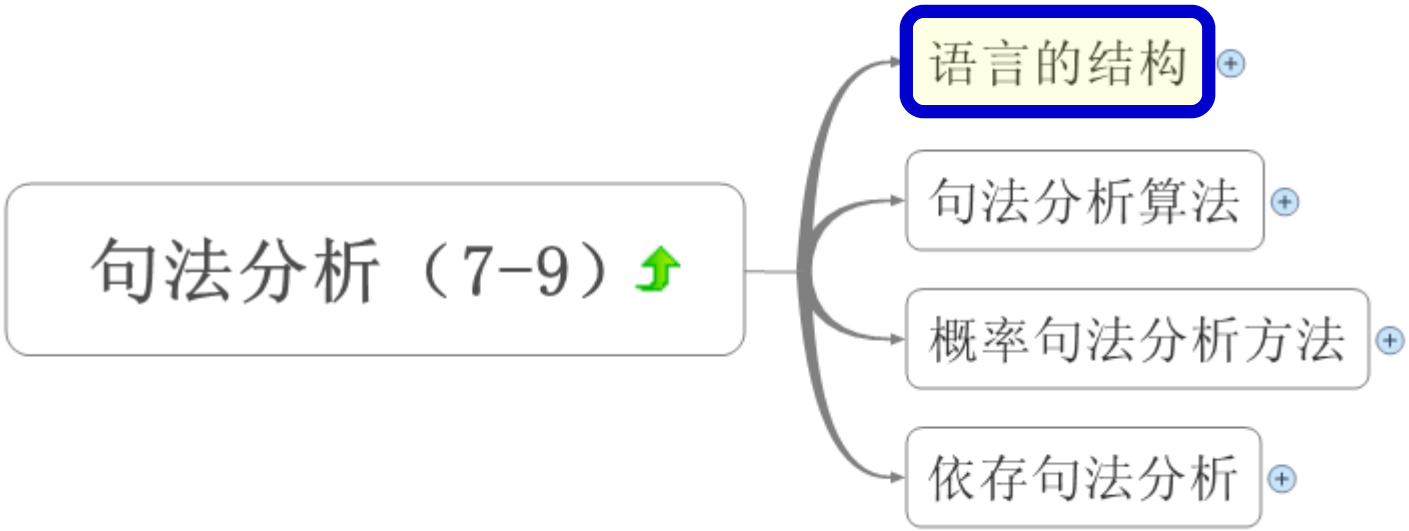
计算语言学

第 7 讲　句法分析（一）

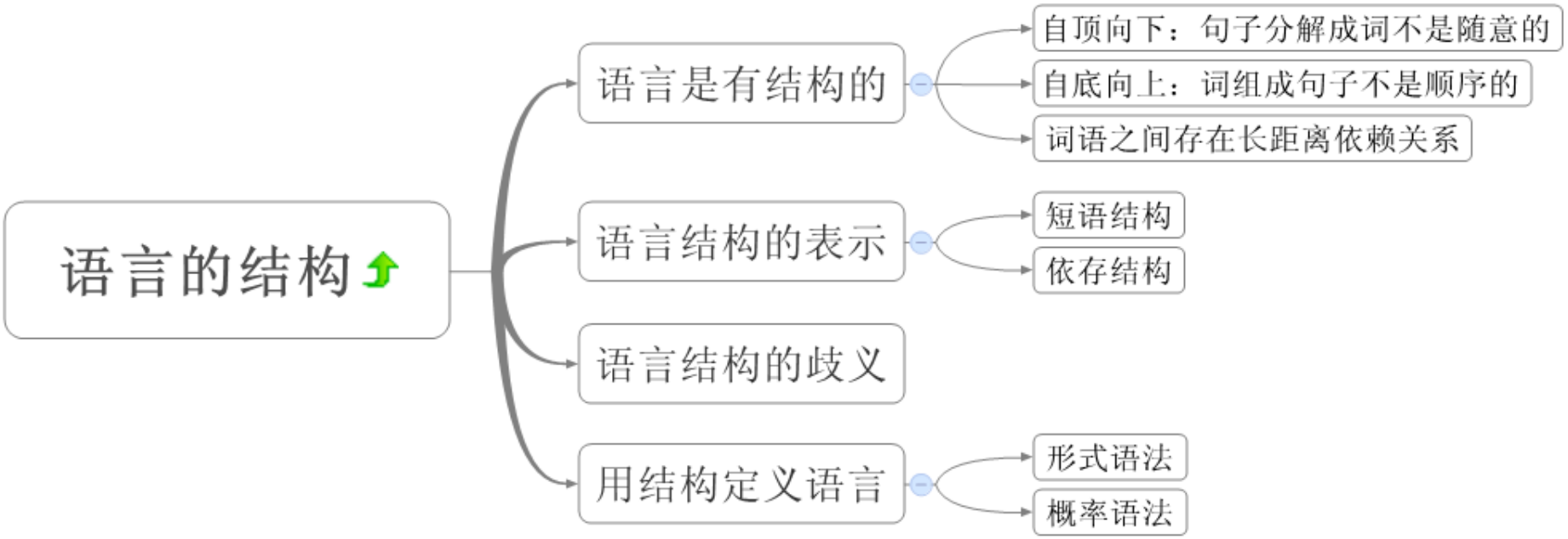
刘群
中国科学院计算技术研究所
liuqun@ict.ac.cn

中国科学院研究生院 2011 年春季课程讲义

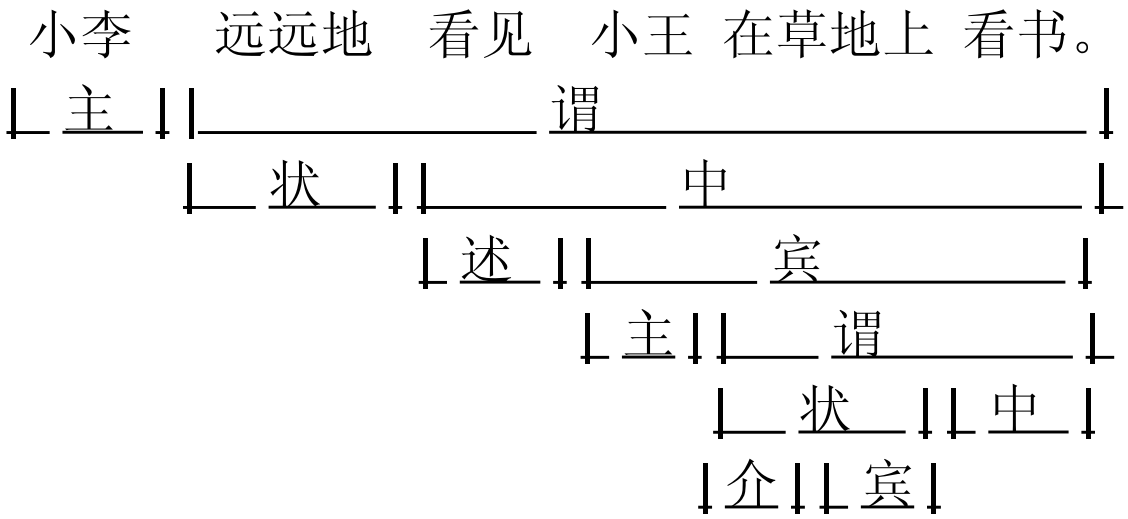
内容提要



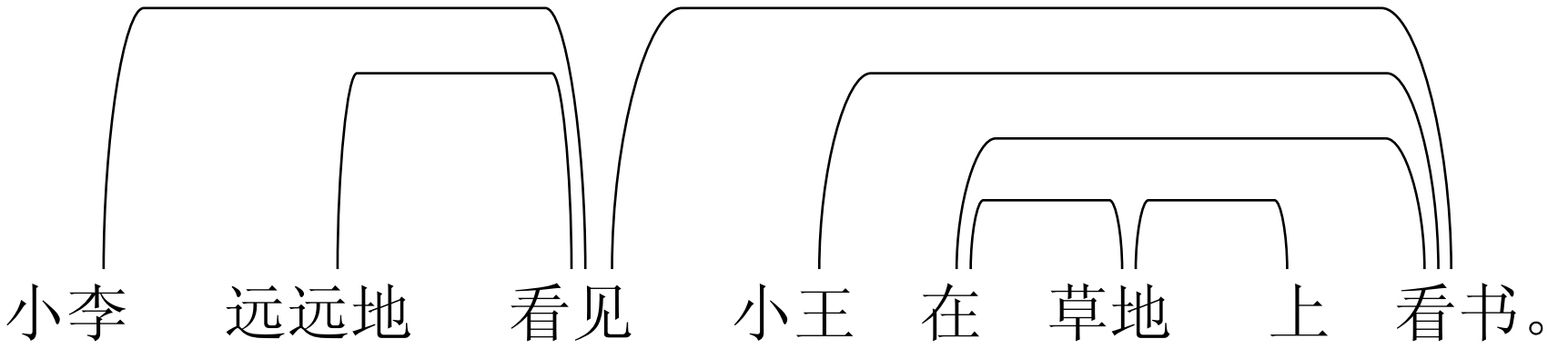
内容提要


语言的结构
语言是有结构的
自顶向下：句子分解成词不是随意的
自底向上：词组成句子不是顺序的
词语之间存在长距离依赖关系
语言结构的表示
短语结构
依存结构
语言结构的歧义
用结构定义语言
形式语法
概率语法

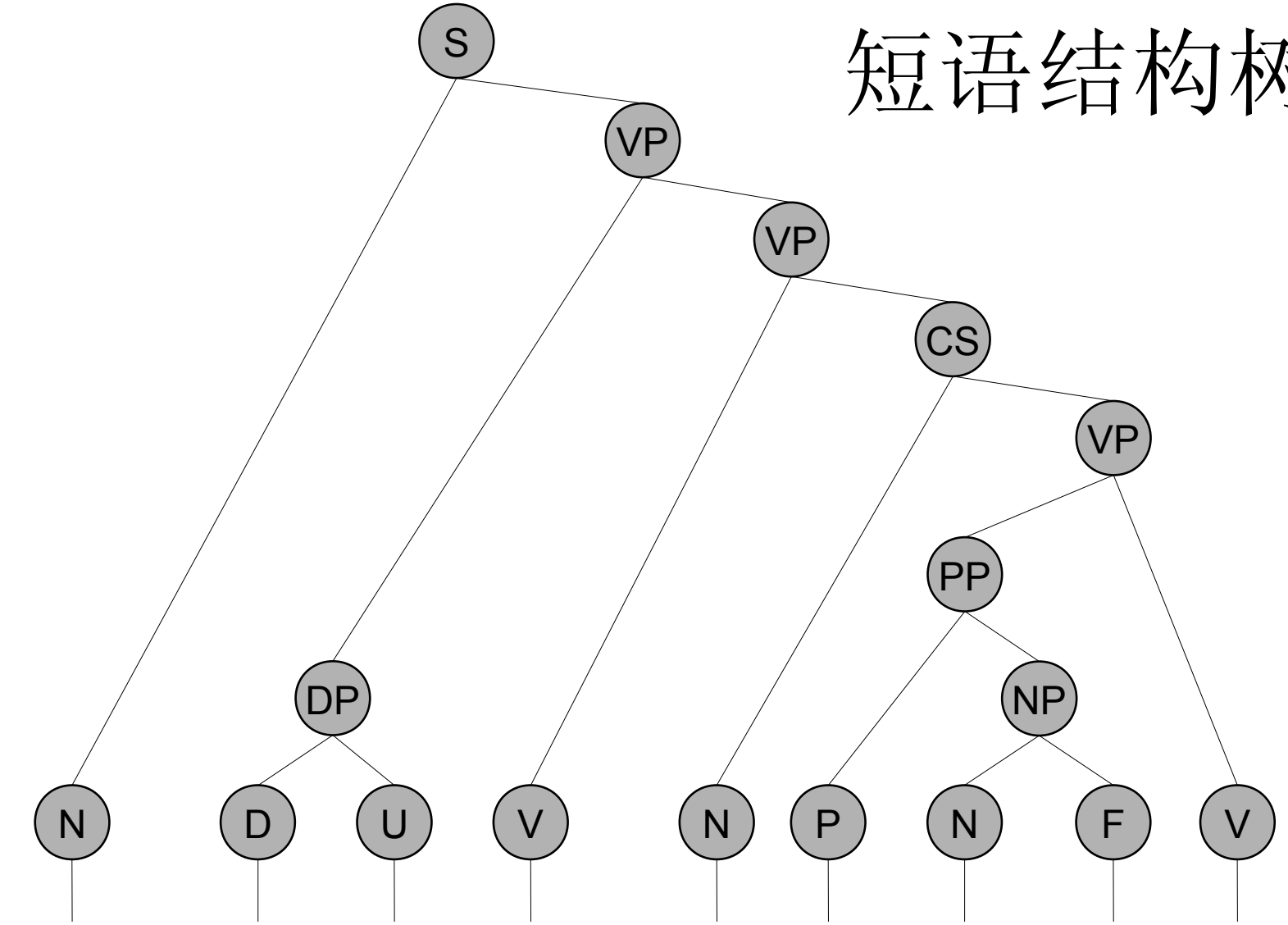
词语到句子的组合顺序



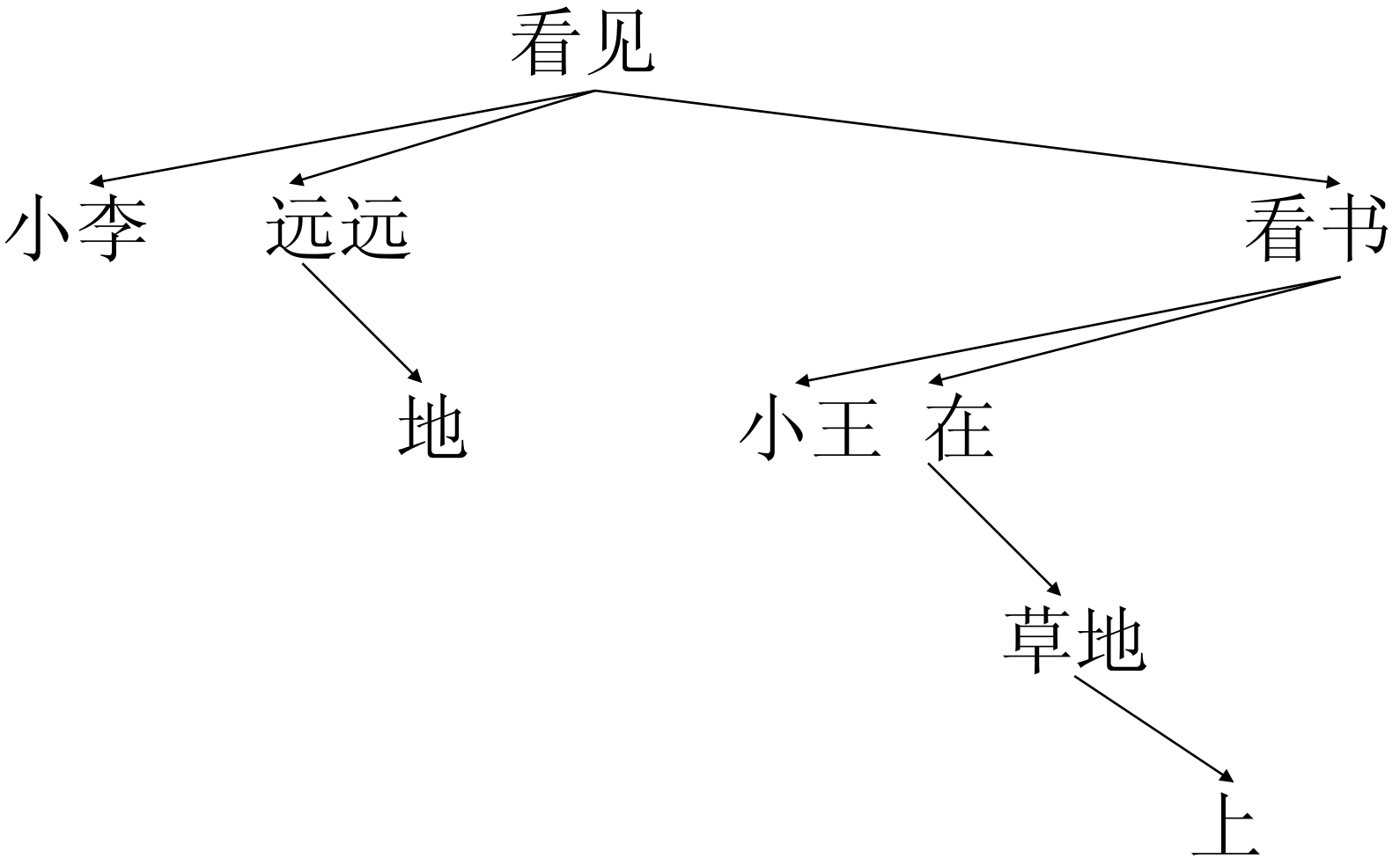
词语之间的依赖关系



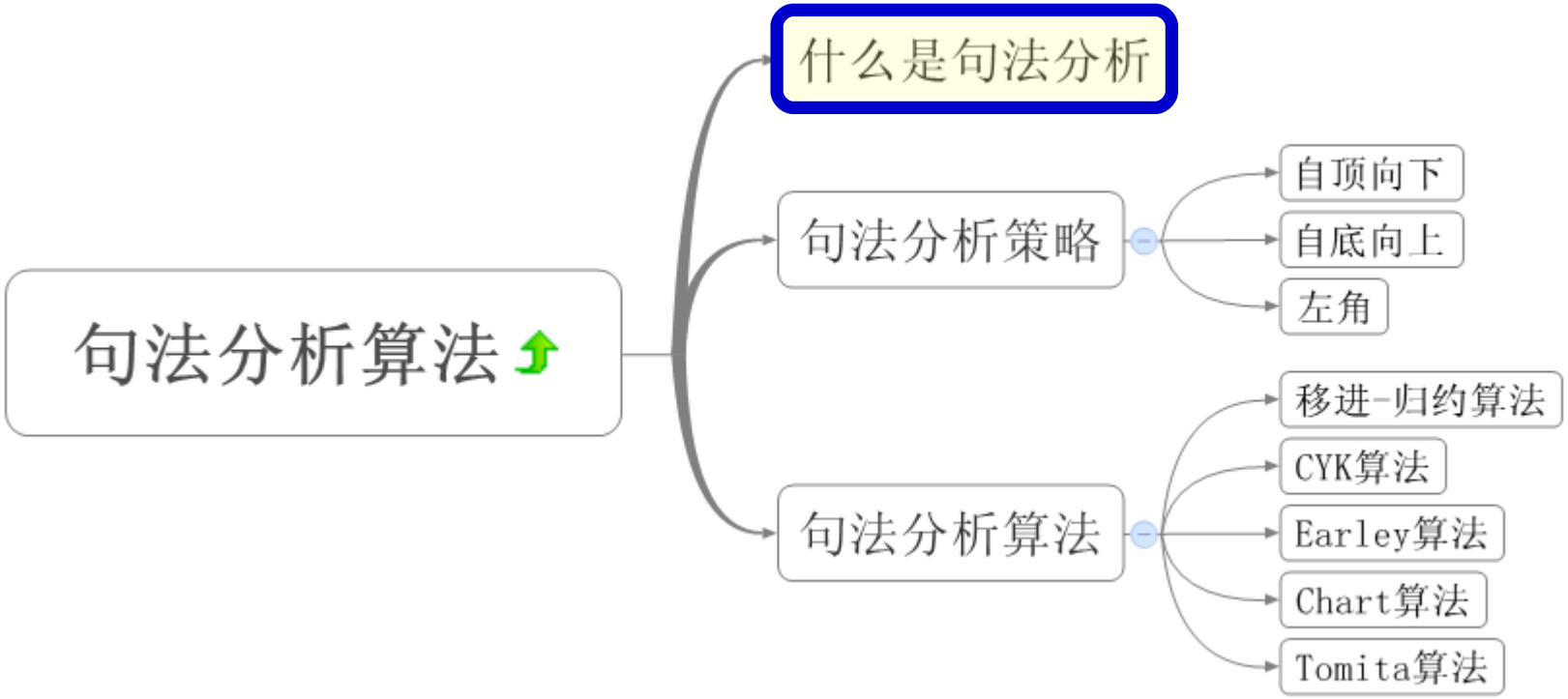
短语结构树



依存结构树



内容提要


句法分析算法
什么是句法分析
句法分析策略
自顶向下
自底向上
左角
句法分析算法
移进-归约算法
CYK算法
Earley算法
Chart算法
Tomita算法

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；
- 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。
- 本讲义将主要介绍上下文无关语法的句法分析器。

与形式语言句法分析的比较

- 形式语言一般是人工构造的语言，是一种确定性的语言，即对于语言中的任何一个句子，只有唯一的一种句法结构是合理的，即使语法本身存在歧义，也往往通过人为的方式规定一种合理的解释。如程序语言中的 if…then if…then…else… 结构，往往都人为规定 else 子句与最接近的 if 子句配对；
- 而在自然语言中，歧义现象是天然地大量存在着的，而且这些歧义的解释往往都有可能是合理的，因此，对歧义现象的处理是自然语言句法分析器最本质的要求。
- 由于要处理大量的歧义现象，导致自然语言句法分析器的复杂程度远高于形式语言的句法分析器。

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

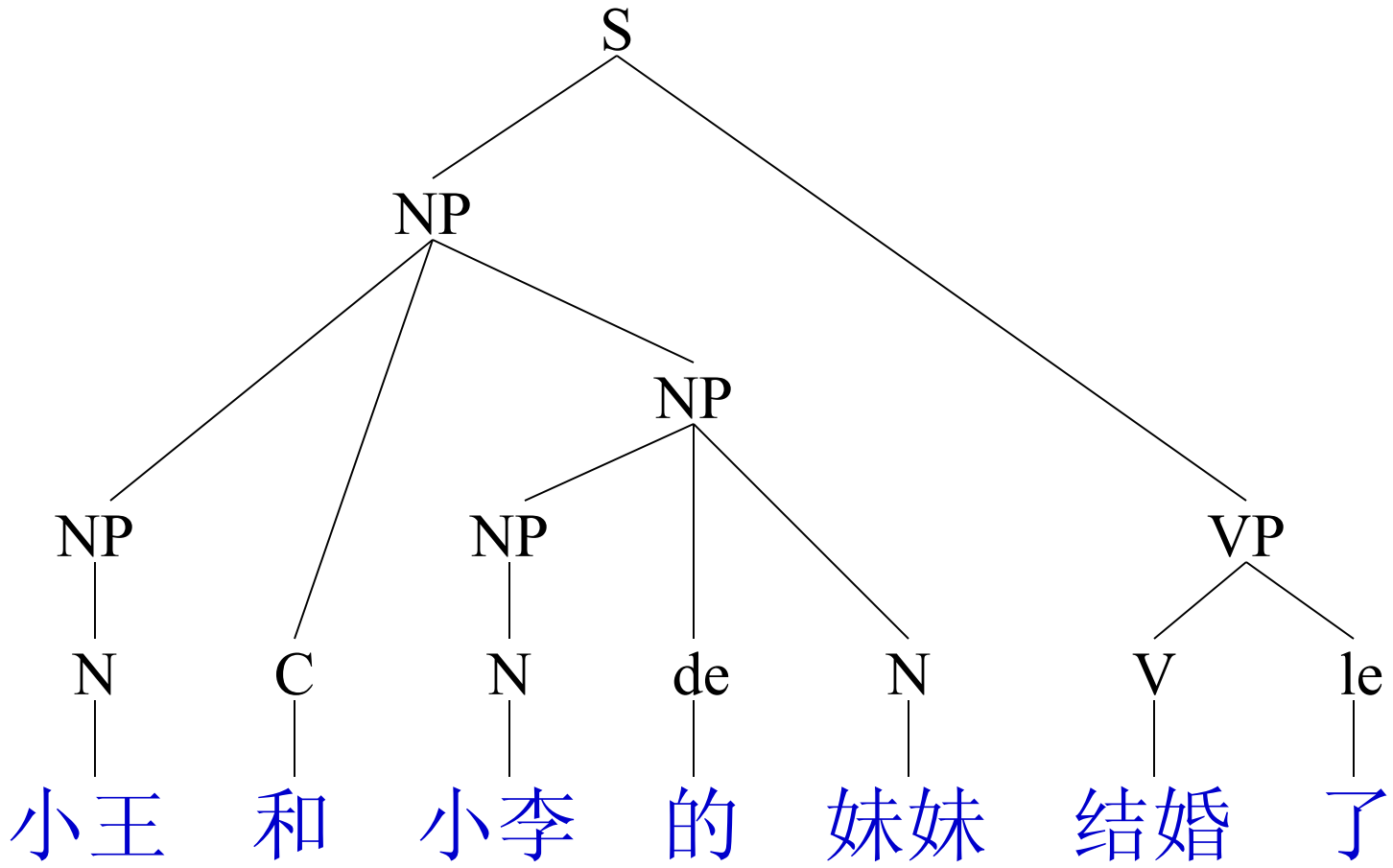
妹妹： N

结婚： V

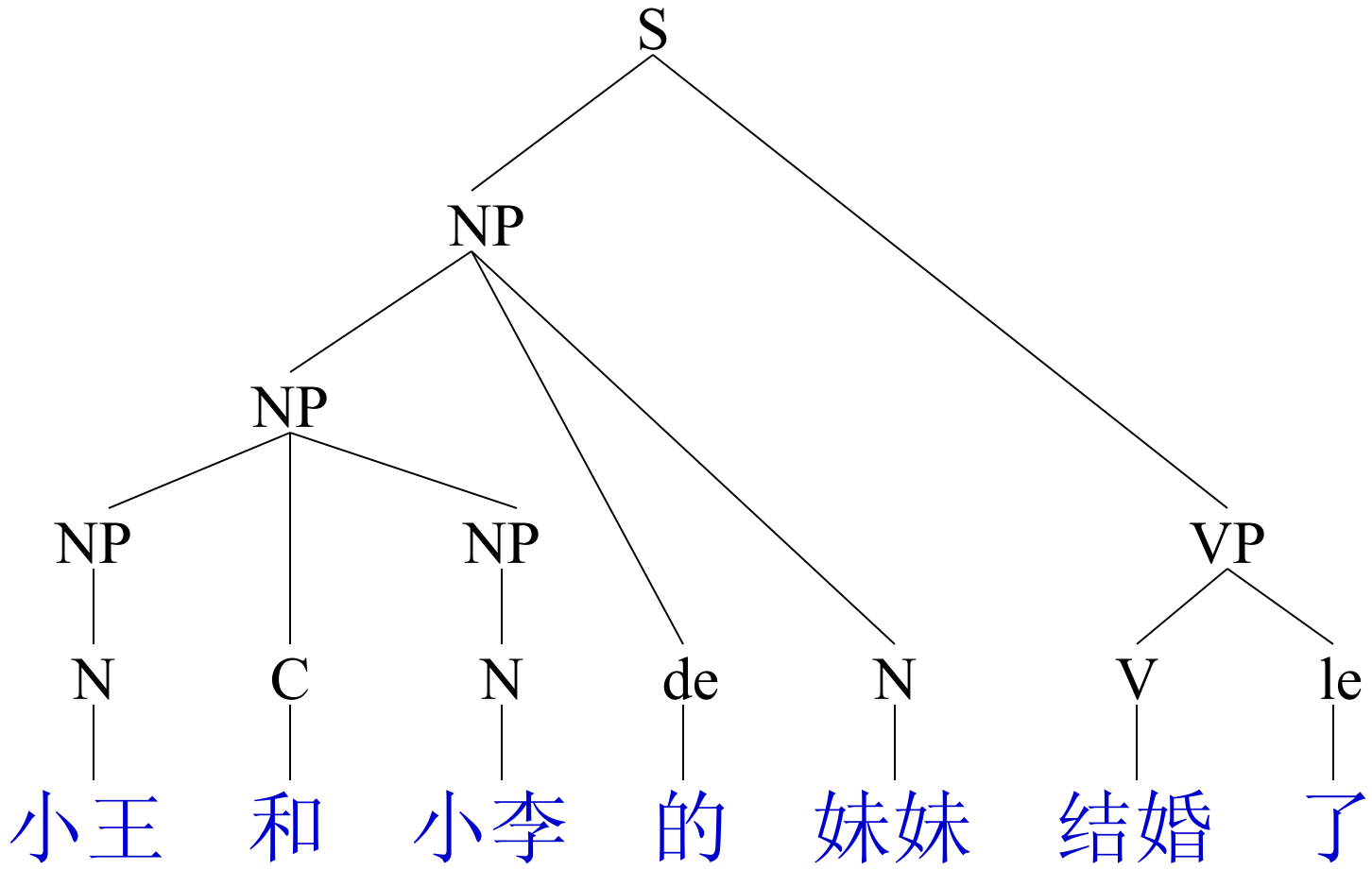
了： le

的： de

例子一分析结果之一



例子一分析结果之二



另一个例子

- 我是县长派来的

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

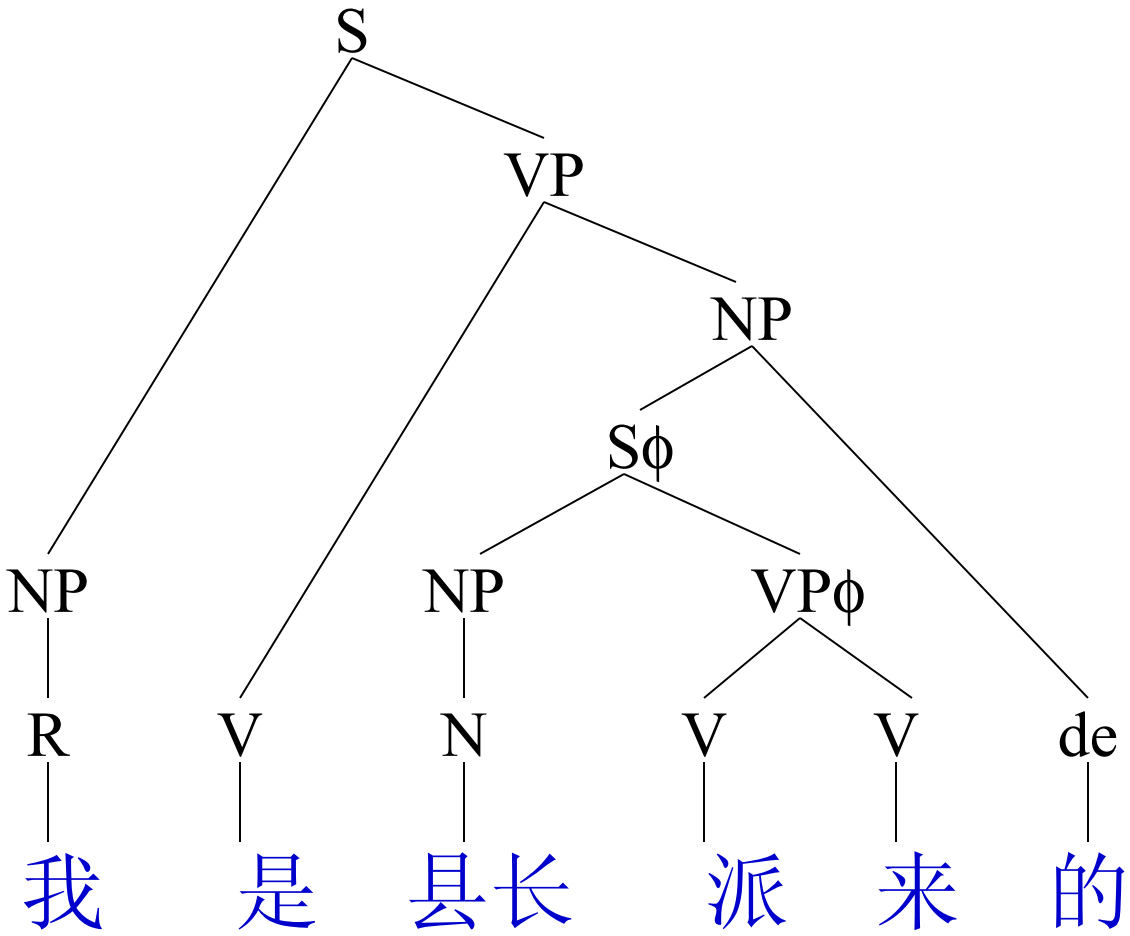
是： V

派： V

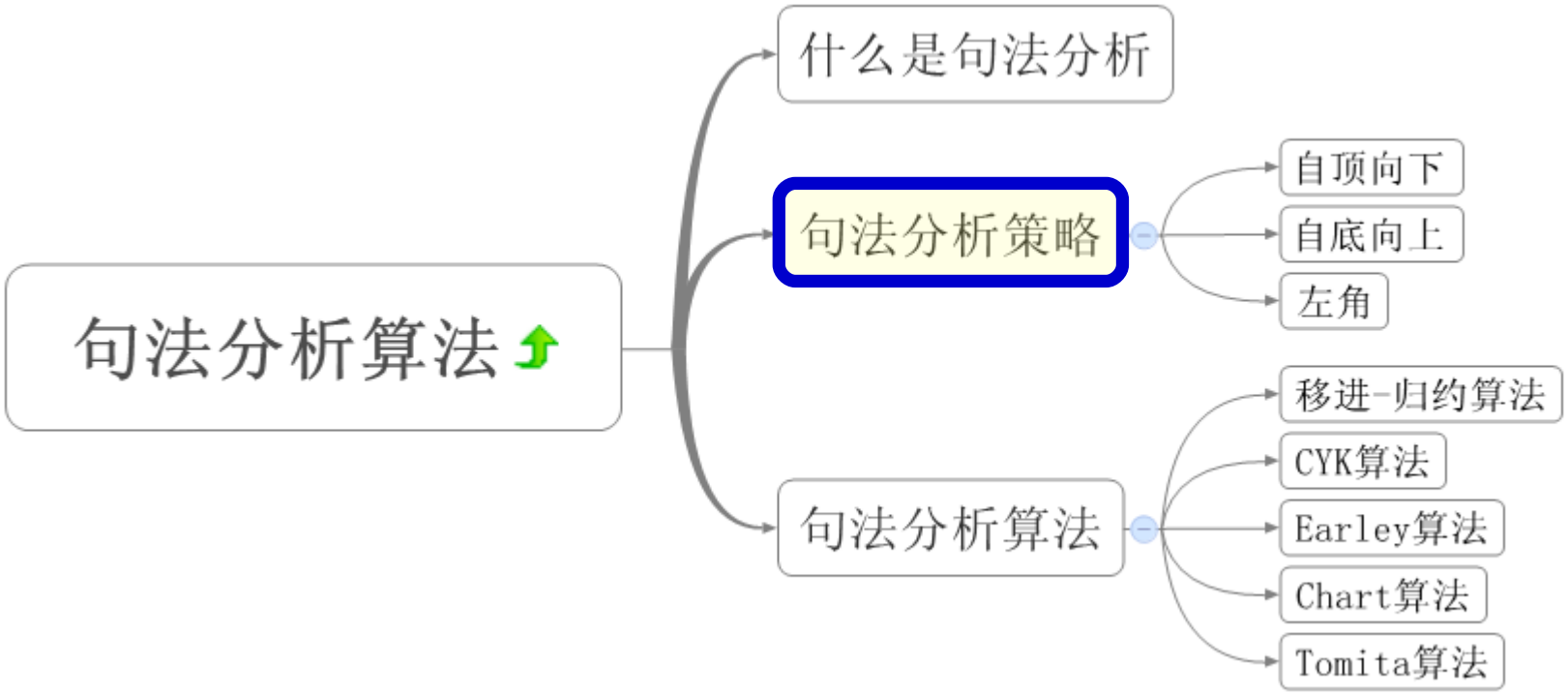
来： V

的： de

另一个例子—分析结果



内容提要


句法分析算法
什么是句法分析
句法分析策略
自顶向下
自底向上
左角
句法分析算法
移进-归约算法
CYK算法
Earley算法
Chart算法
Tomita算法

句法分析的基本策略

句法分析通常采用的策略有：

- 自顶向下分析法；
- 自底向上分析法；
- 左角分析法；
- 其他策略。

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）；

自顶向下和自底向上分析法 (1)

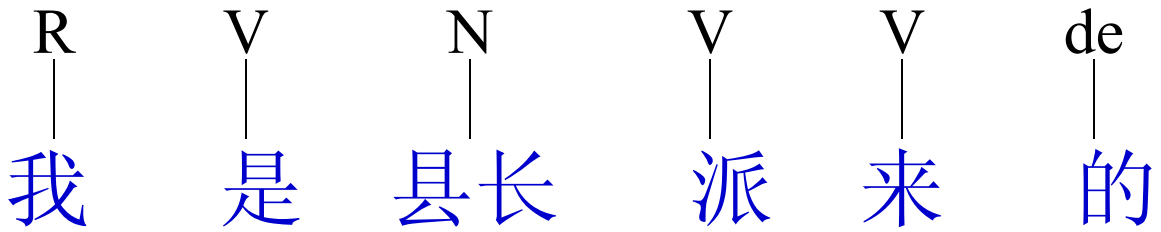
- 句法分析的过程也可以理解为句法树的构造过程
- 所谓自顶向下分析法也就是先构造句法树的根结点，再逐步向下扩展，直到叶结点；
- 所谓自底向上分析法也就是先构造句法树的叶结点，再逐步向上合并，直到根结点。

自顶向下和自底向上分析法 (2)

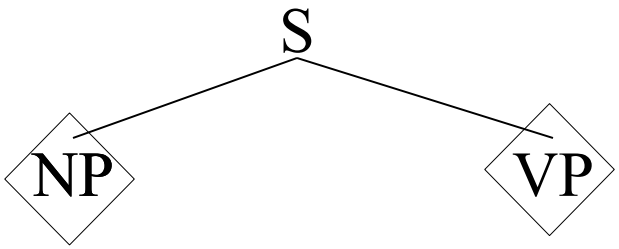
- 自顶向下的方法又称为基于预测的方法，也就是说，这种方法是先产生对后面将要出现的成分的预期，然后再通过逐步吃进待分析的字符串来验证预期。如果预期得到了证明，就说明待分析的字符串可以被分析为所预期的句法结构。如果某一个环节上预期出了差错，那就要用另外的预期来替换（即回溯）。如果所有环节上所有可能的预期都被吃进的待分析字符串所“反驳”，那就说明待分析的字符串不可能是一个合法的句子，分析失败。
- 自底向上的方法也叫基于归约的方法。就是说，这种方法是先逐步吃进待分析字符串，把它们从局部到整体层层归约为可能的成分。如果整个待分析字符串被归约为开始符号 S，那么分析成功。如果在某个局部证明不可能有任何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。

自顶向下分析法—示例 (1)

查词典



自顶向下分析法—示例 (2)

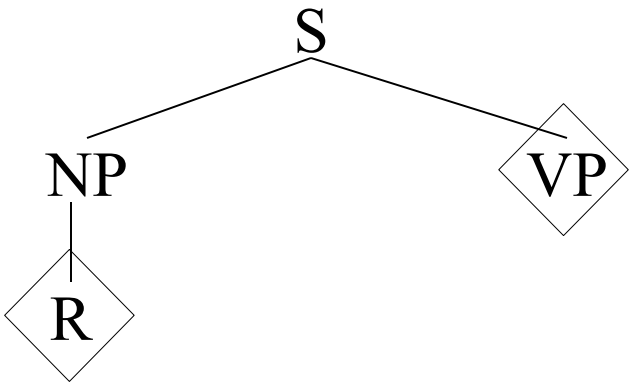


使用规则：

S → NP VP

R	V	N	V	V	de
我	是	县长	派	来	的

自顶向下分析法—示例 (3)

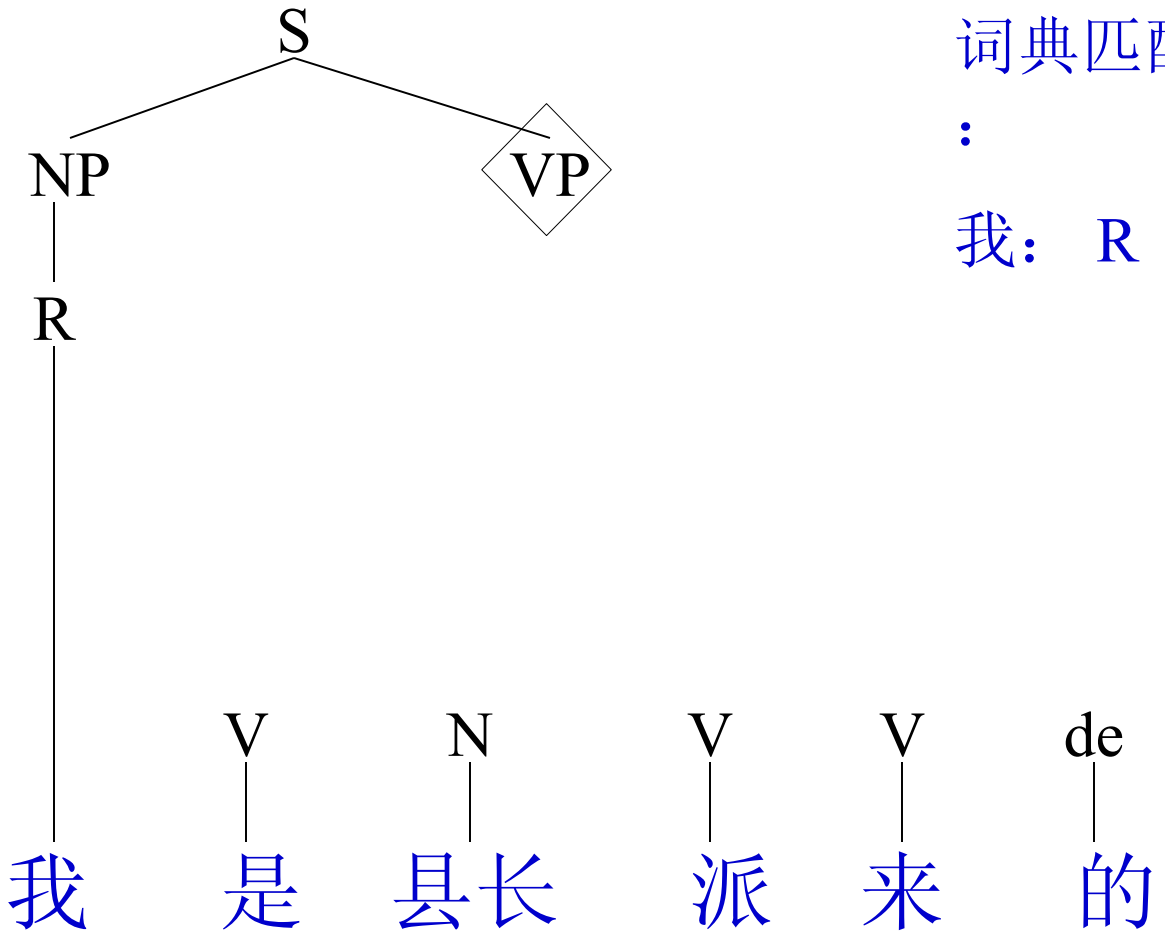


使用规则：

NP→R

R 我　V 是　N 县长　V 派　V 来　de 的

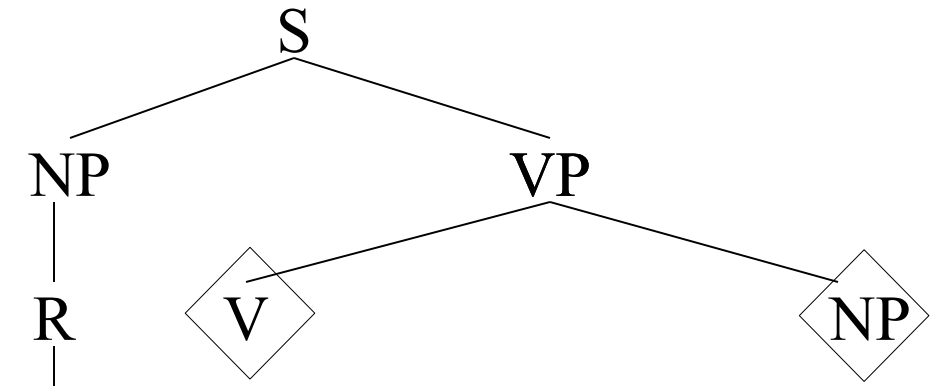
自顶向下分析法—示例 (4)



词典匹配成功：

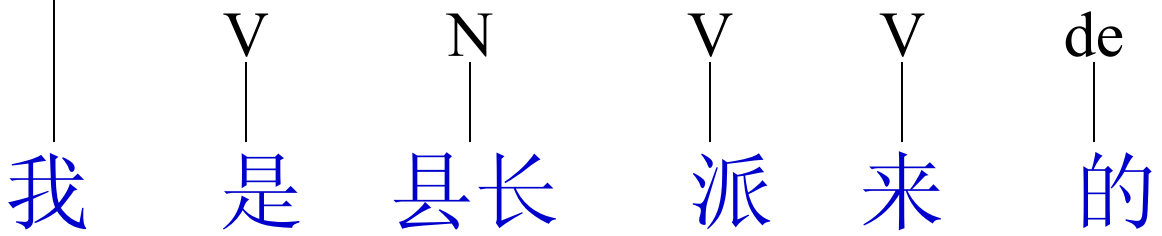
我： R

自顶向下分析法—示例 (5)

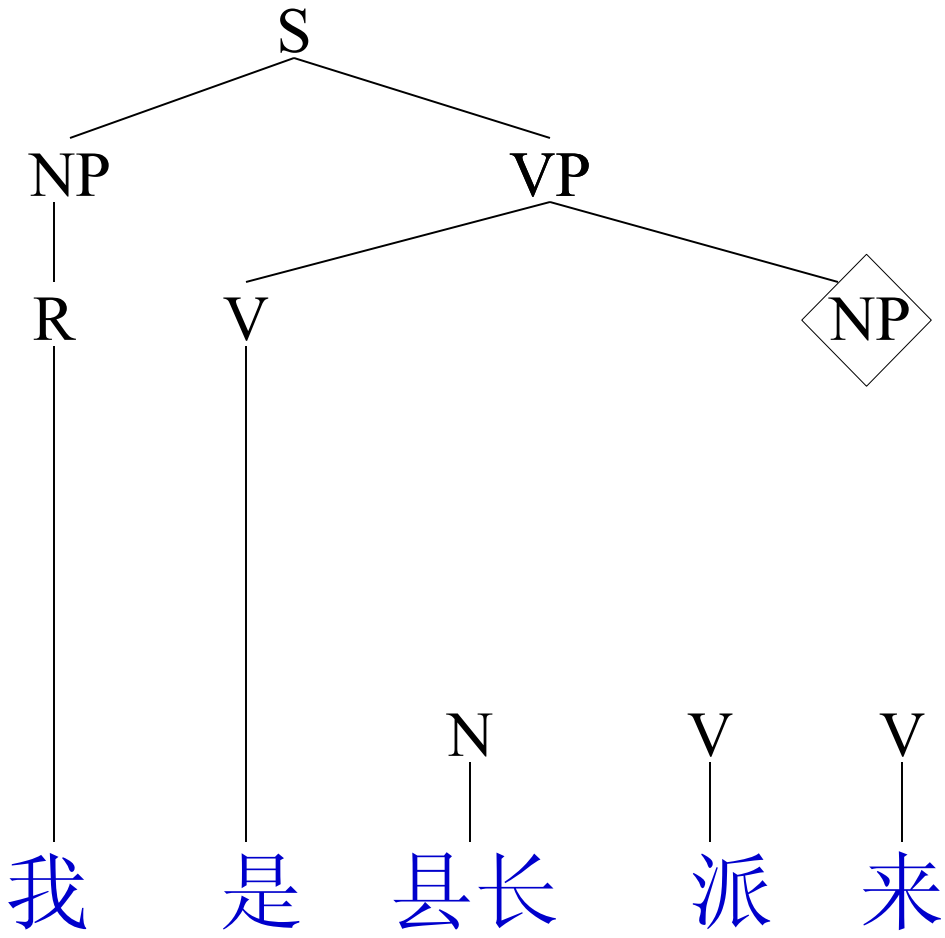


使用规则：

VP→V NP



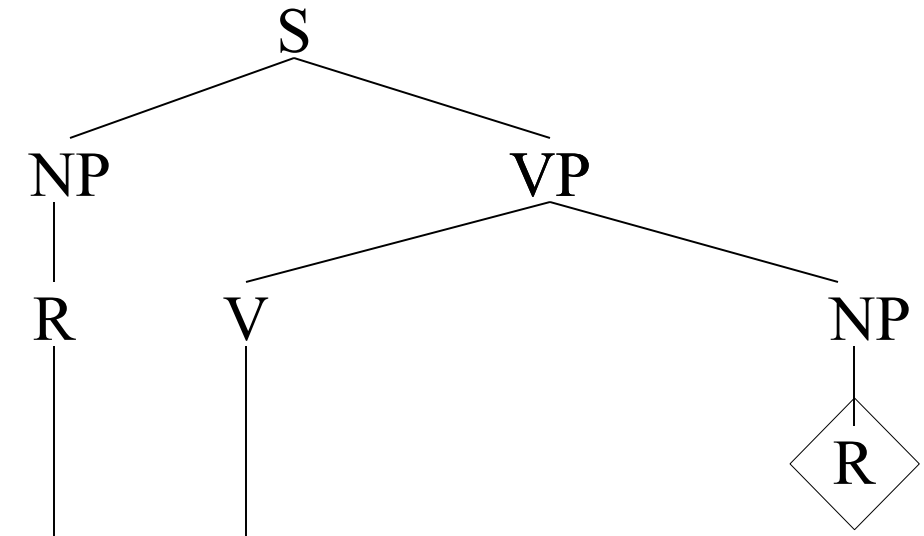
自顶向下分析法—示例 (6)



词典匹配成功：

是： V

自顶向下分析法—示例 (7)



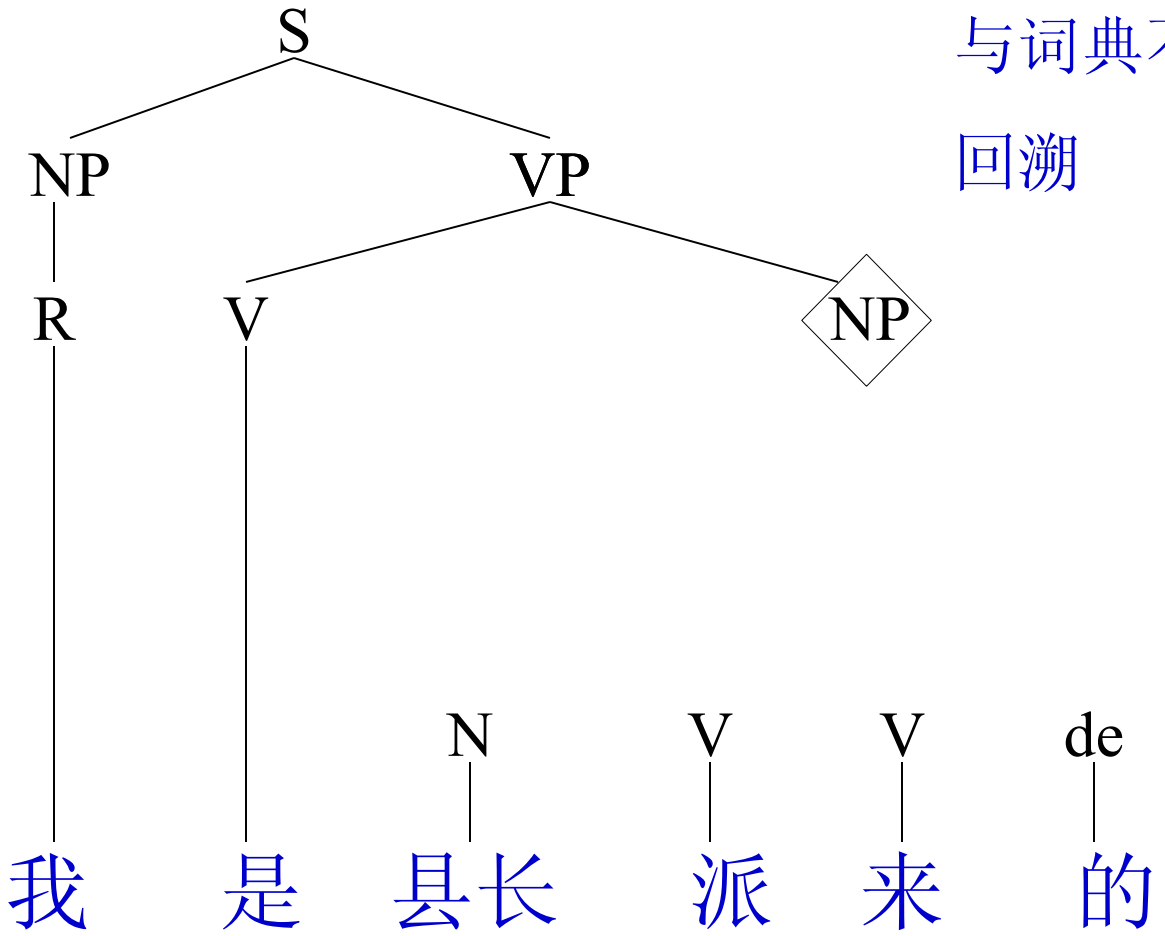
使用规则：

NP→R

N V V de

我 是 县长 派 来 的

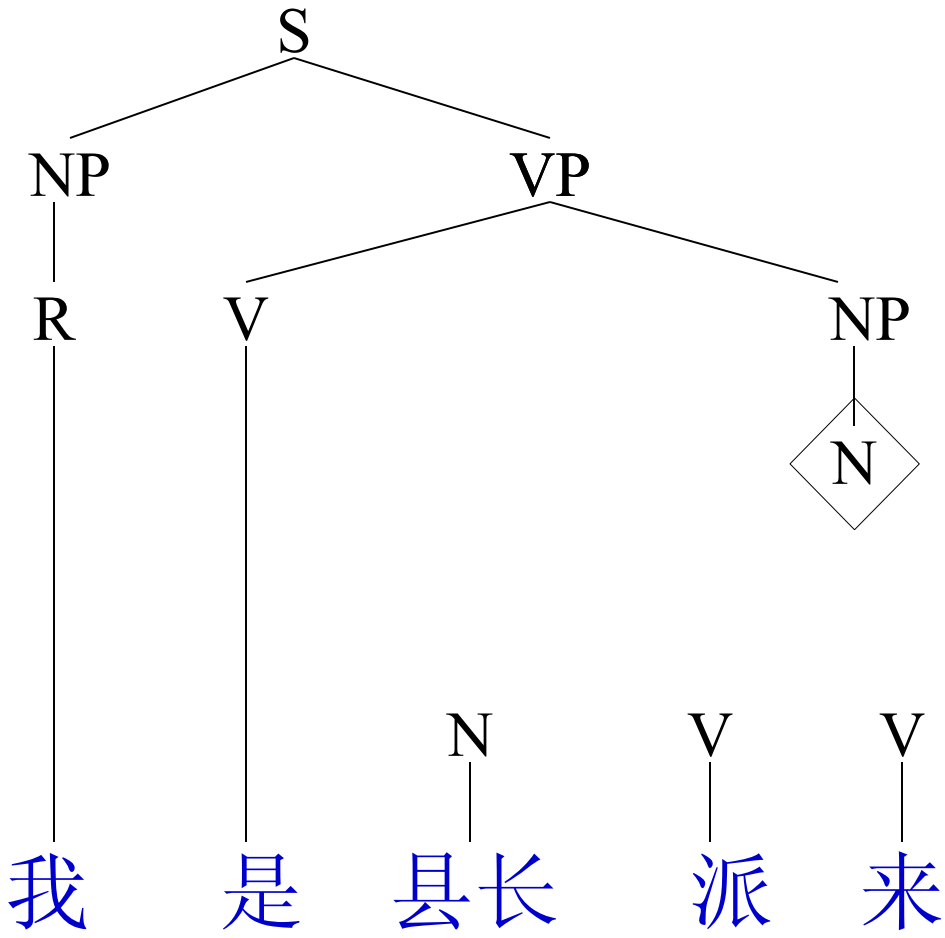
自顶向下分析法—示例 (8)



与词典不匹配

回溯

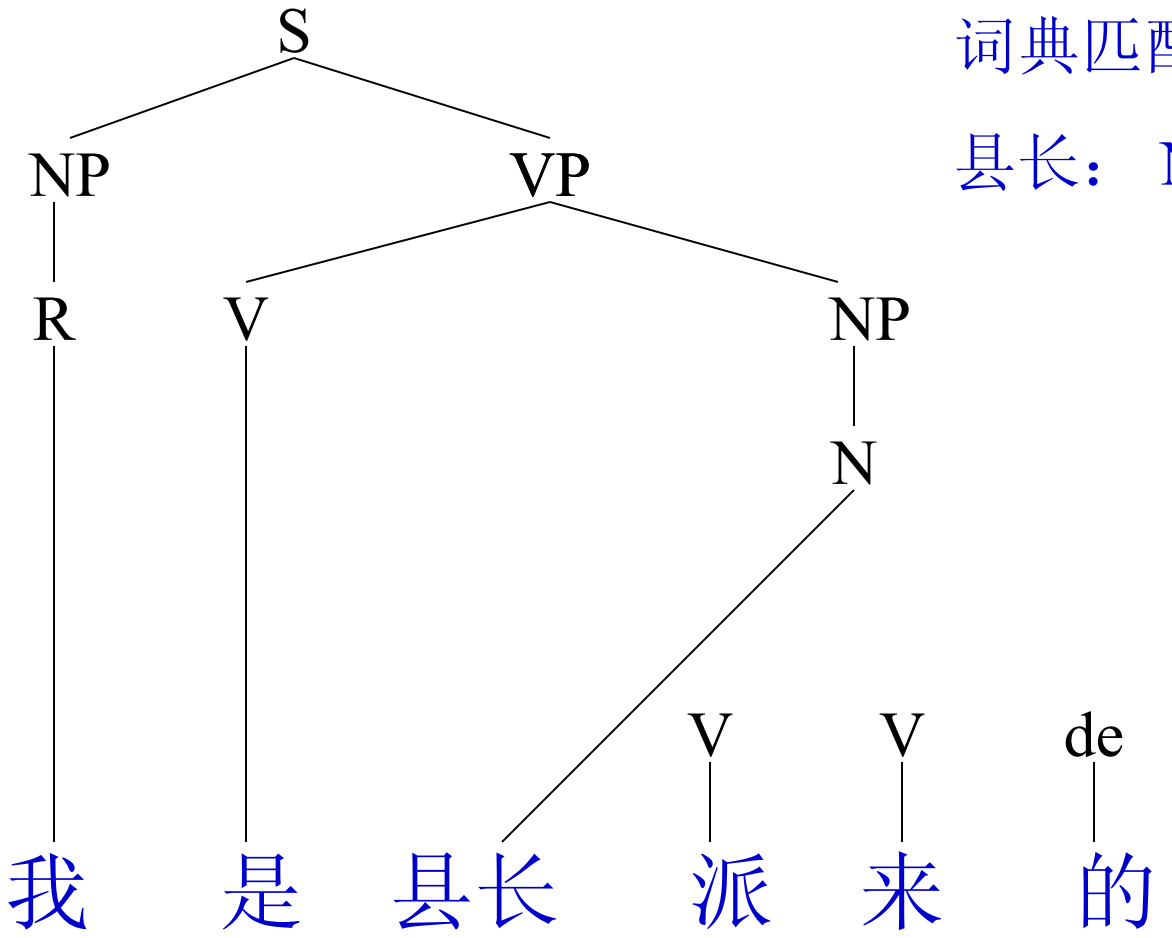
自顶向下分析法—示例 (9)



使用下一条规则：

NP→N

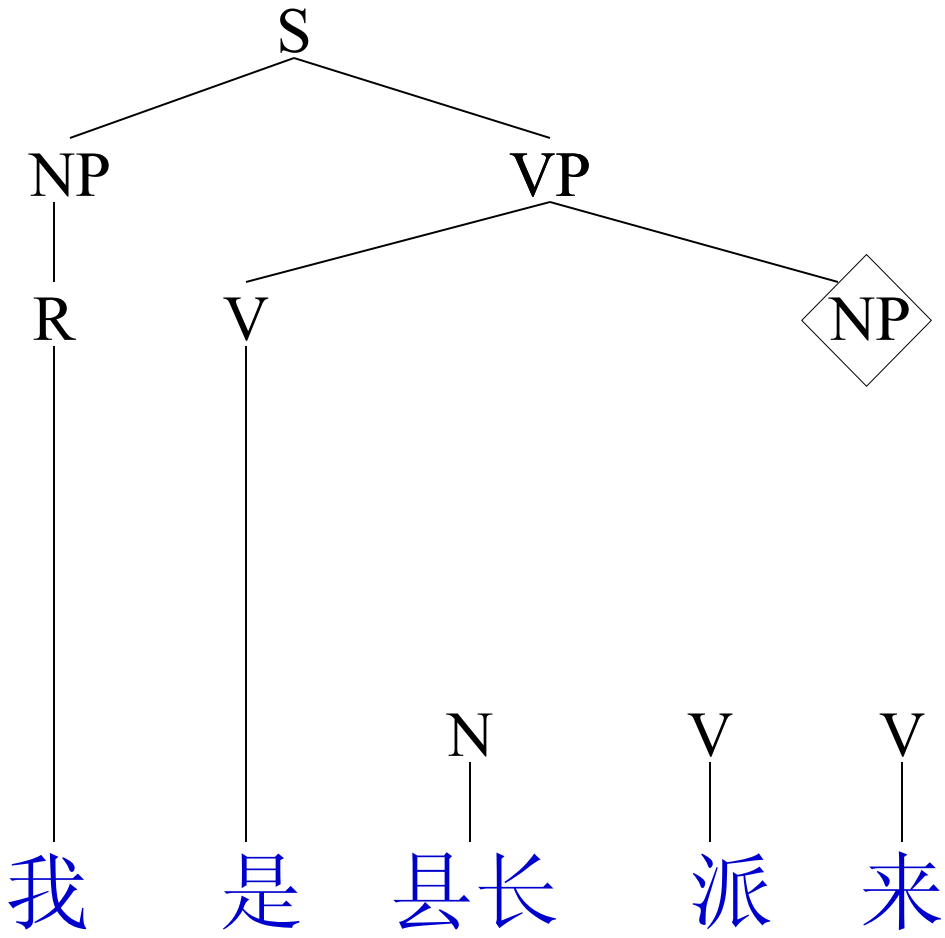
自顶向下分析法—示例 (10)



词典匹配成功：

县长： N

自顶向下分析法—示例 (11)

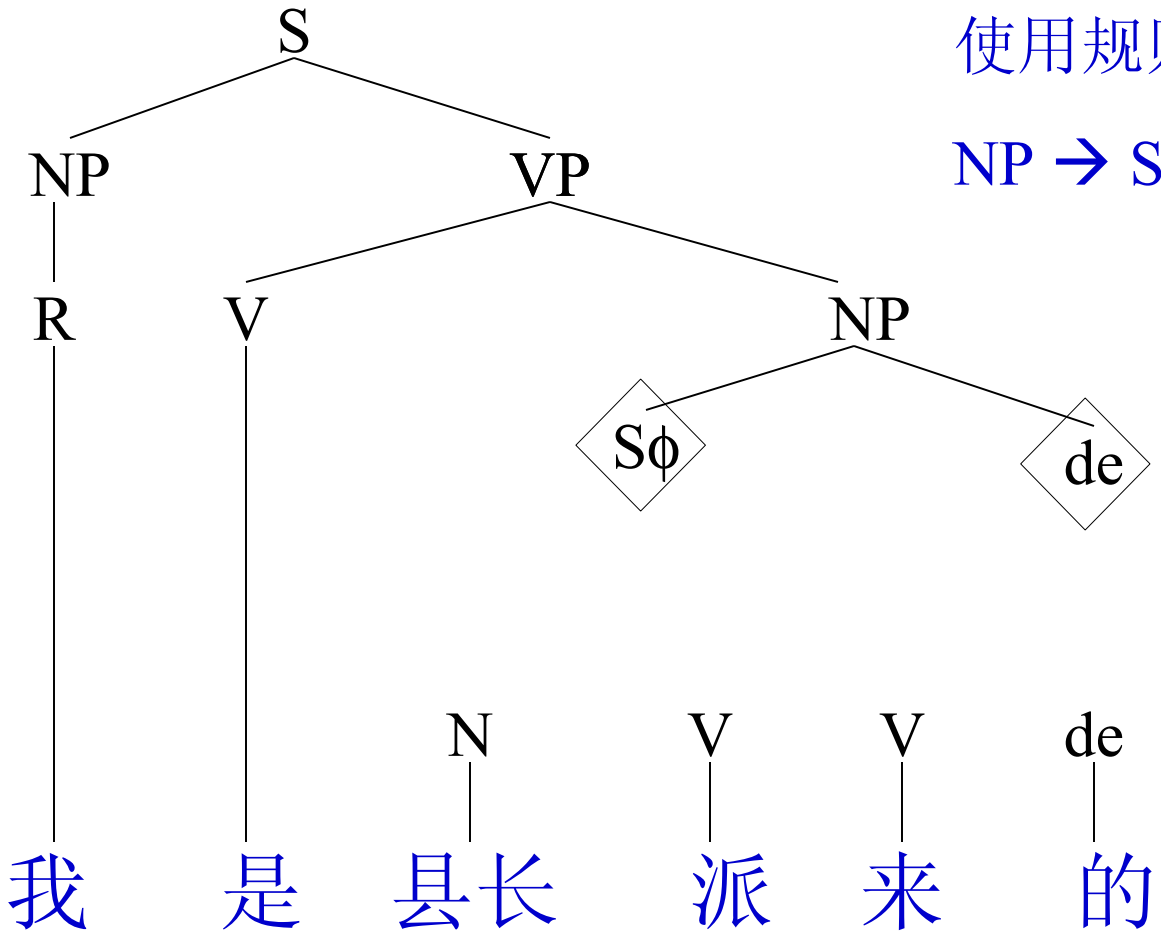


句法树扩展完毕，

但句子没有完全

匹配成功，回溯

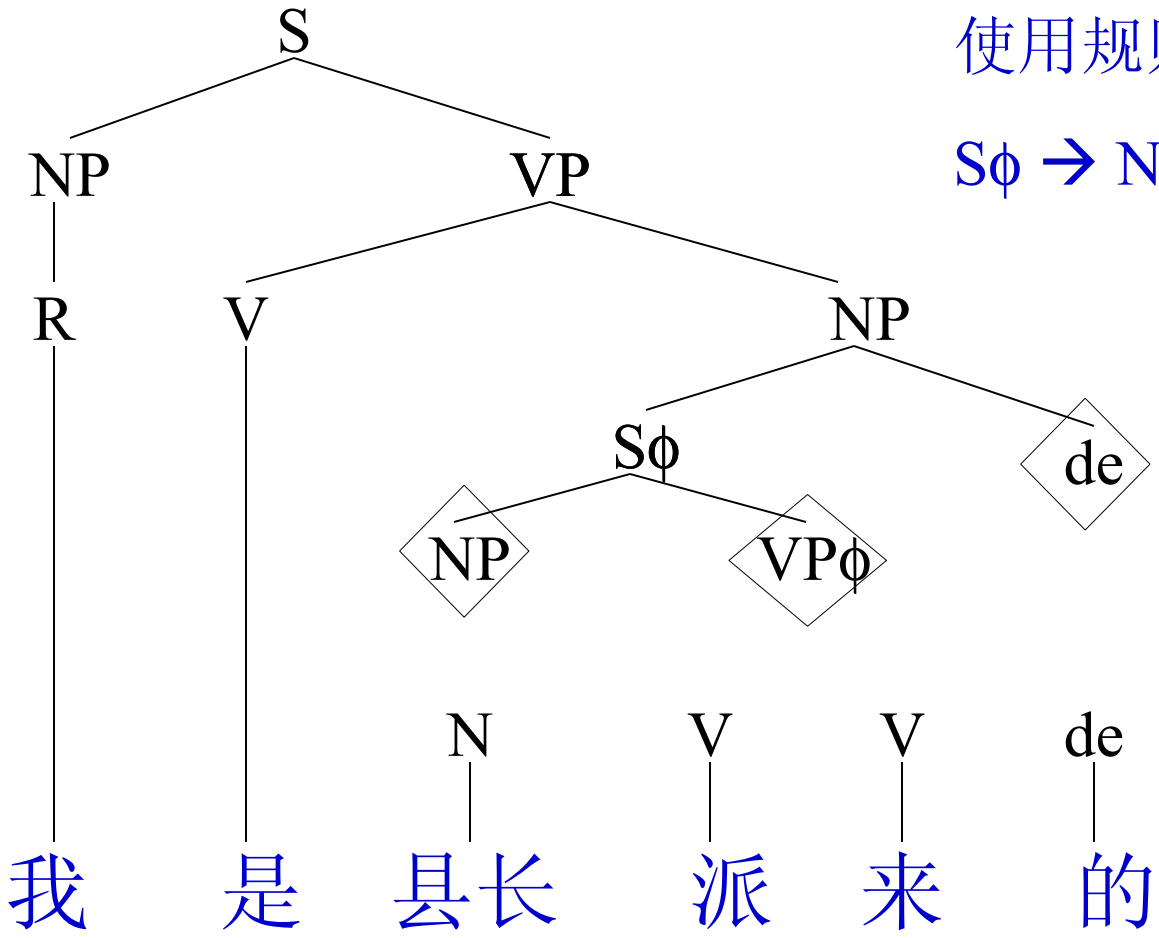
自顶向下分析法—示例 (12)



使用规则：

NP → Sϕ de

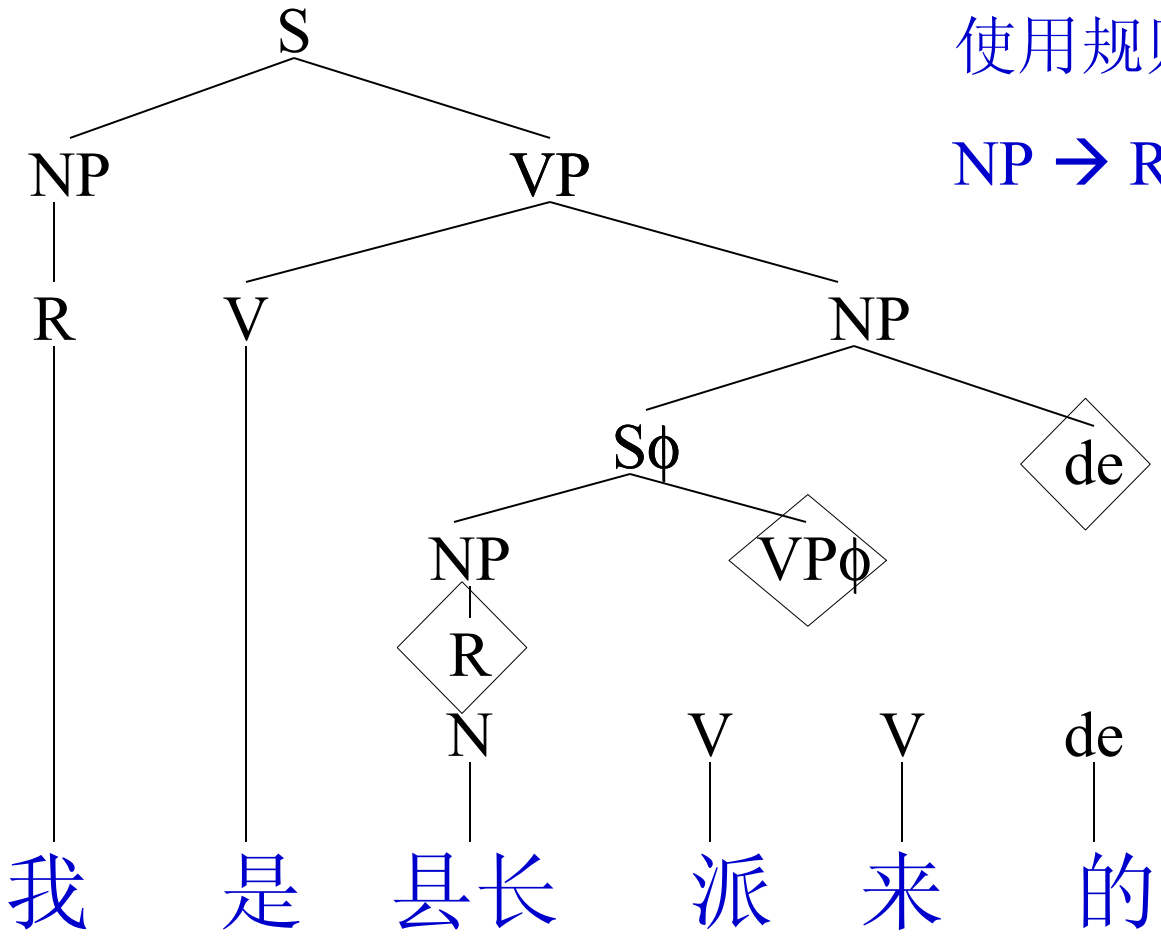
自顶向下分析法—示例 (13)



使用规则：

Sϕ → NP Vϕ

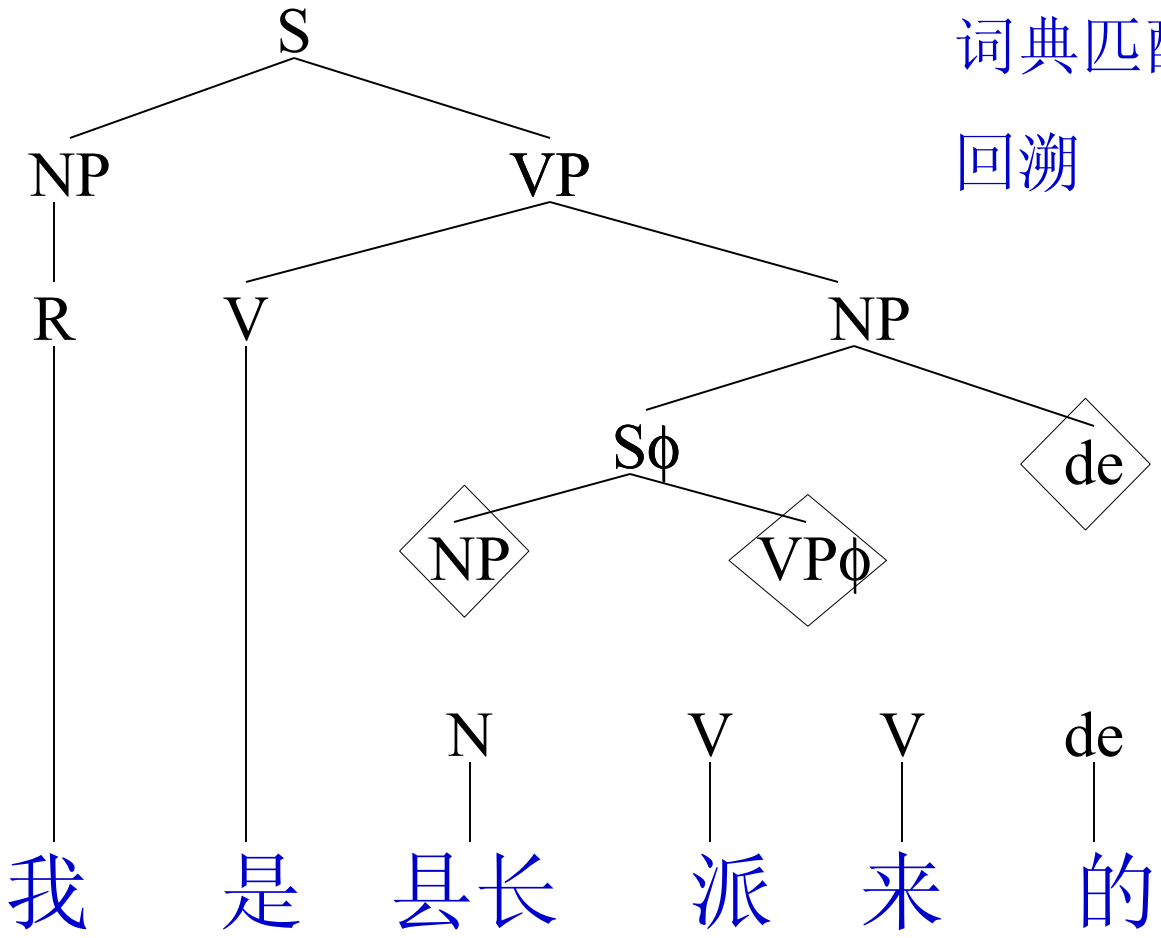
自顶向下分析法—示例 (14)



使用规则：

NP → R

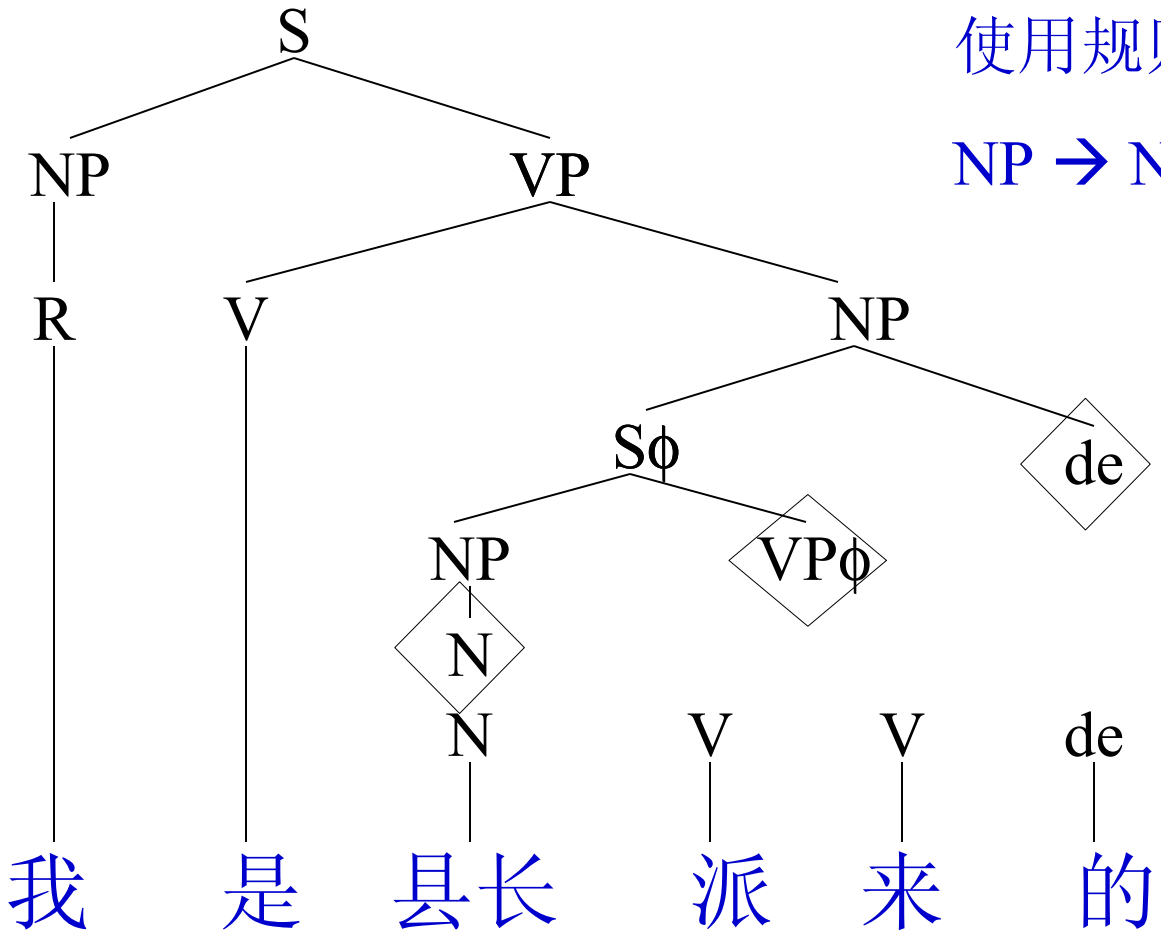
自顶向下分析法一示例 (15)



词典匹配失败

回溯

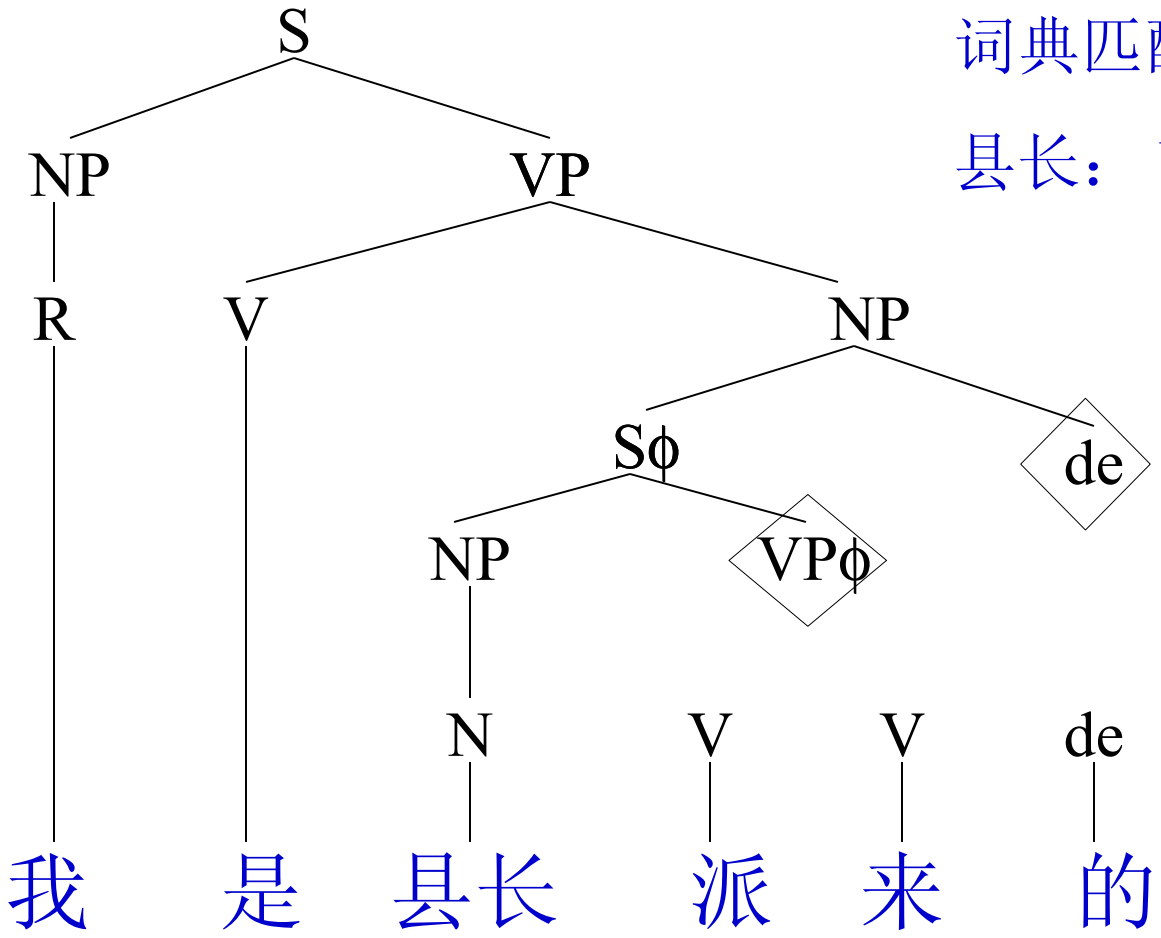
自顶向下分析法—示例 (16)



使用规则：

NP → N

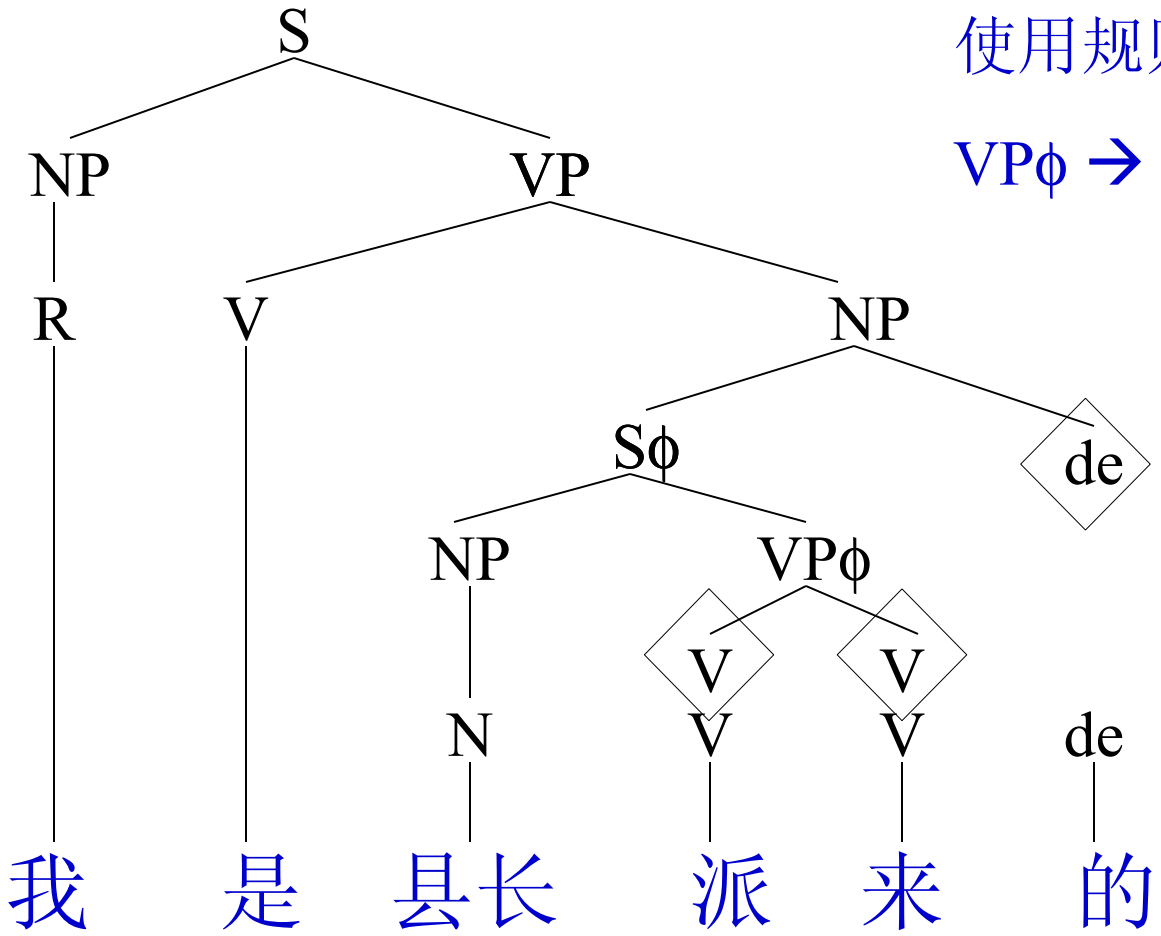
自顶向下分析法—示例 (17)



词典匹配成功：

县长： N

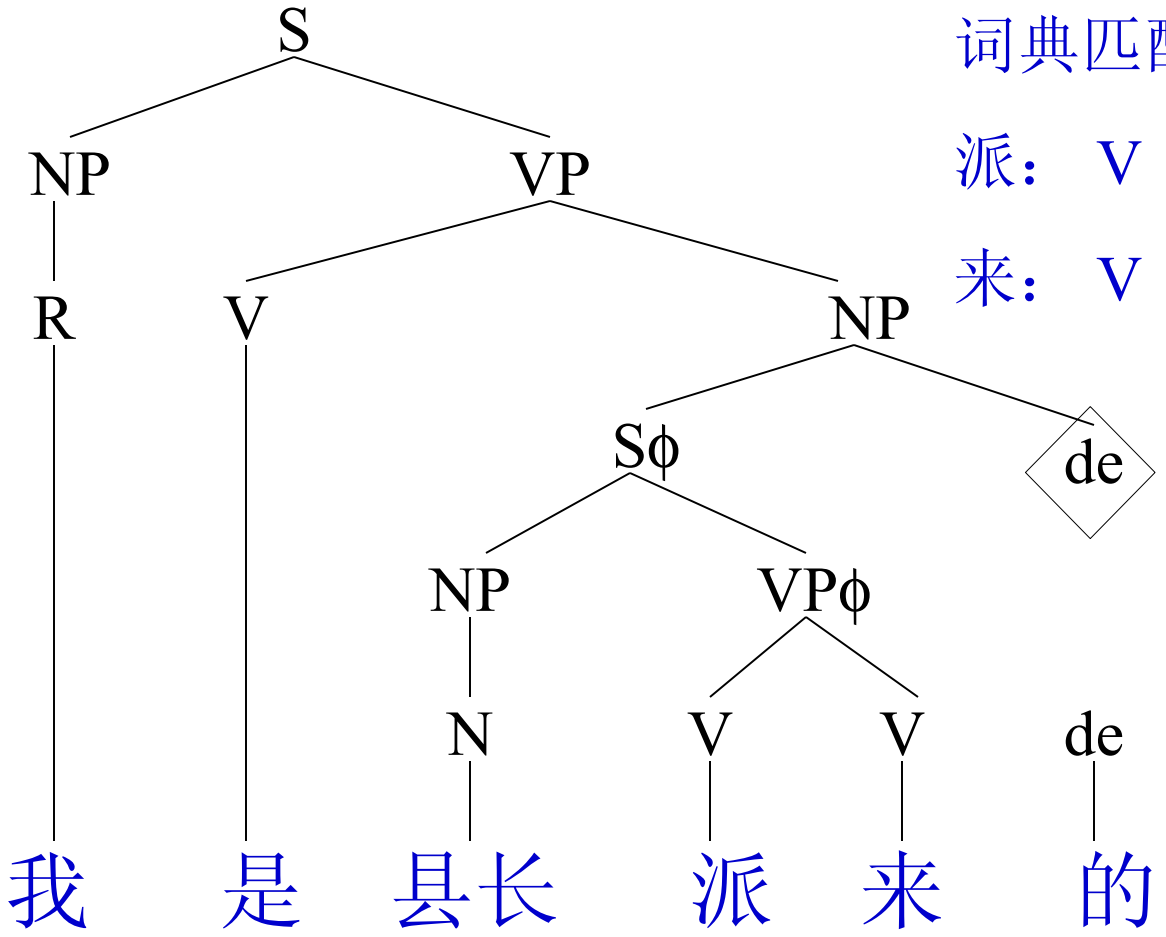
自顶向下分析法—示例 (18)



使用规则：

VPϕ → V V

自顶向下分析法—示例 (19)

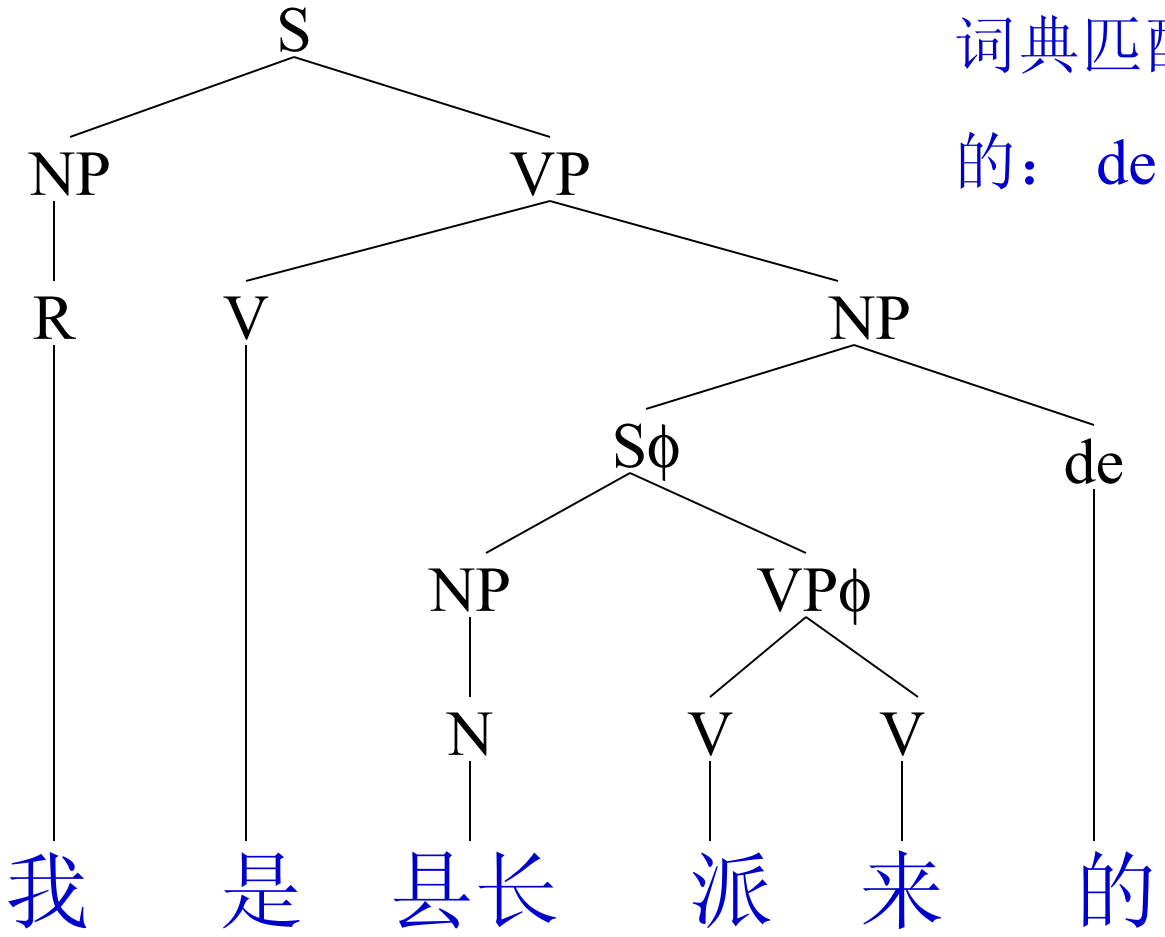


词典匹配成功：

派： V

来： V

自顶向下分析法一示例 (20)



词典匹配成功：

的： de

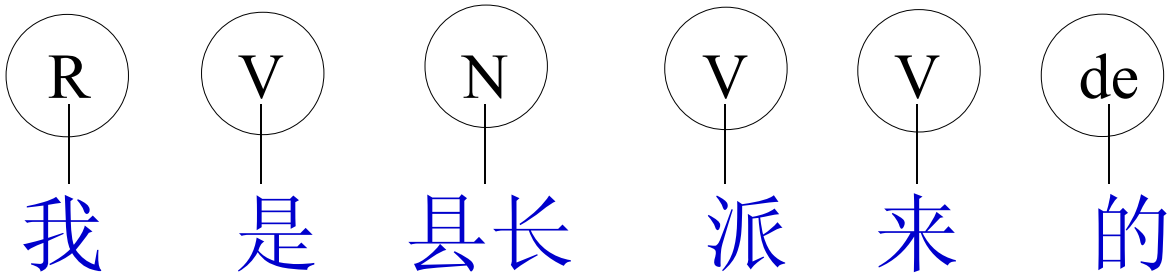
句法树扩展完毕

恰好句子匹配完成

分析成功

自底向上分析法—示例 (1)

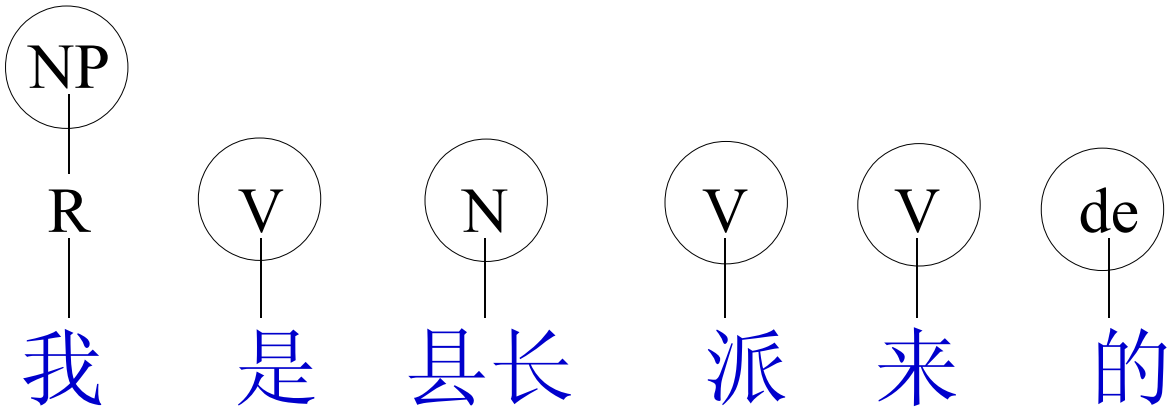
查词典



自底向上分析法—示例 (2)

使用规则：

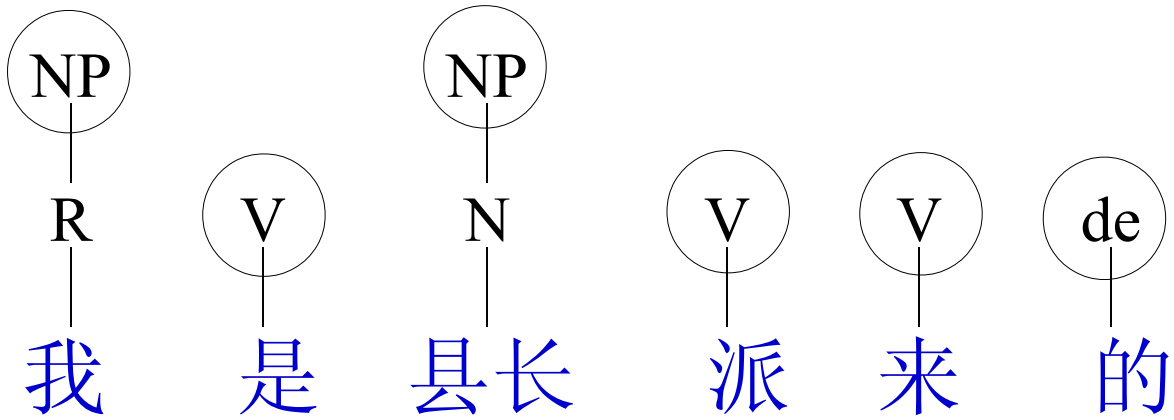
NP→R



自底向上分析法—示例 (3)

使用规则：

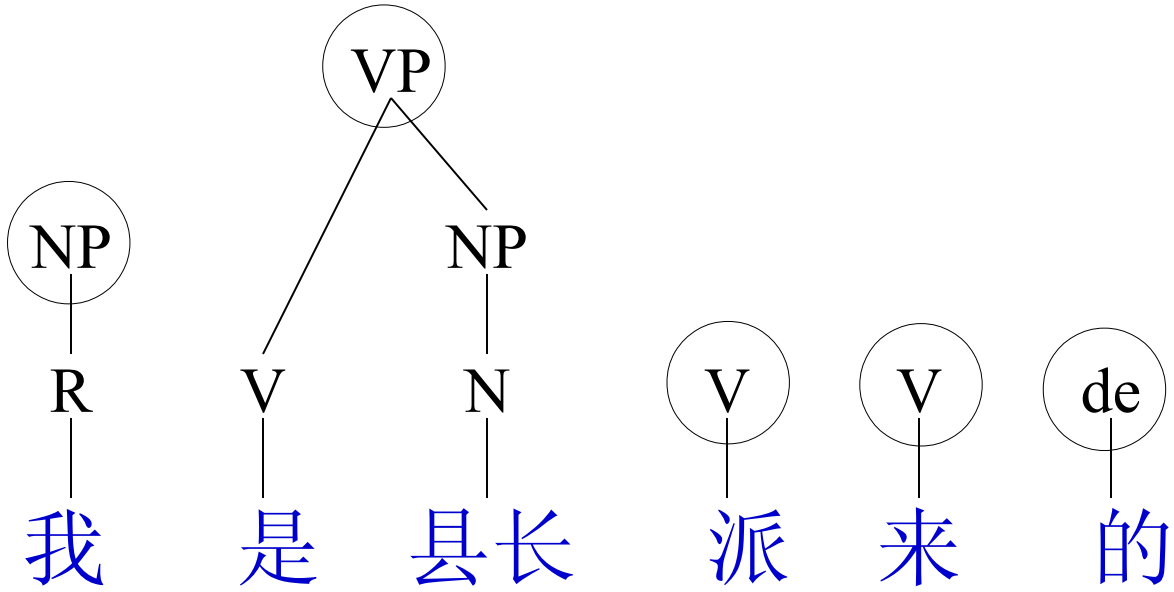
NP→N



自底向上分析法—示例 (4)

使用规则：

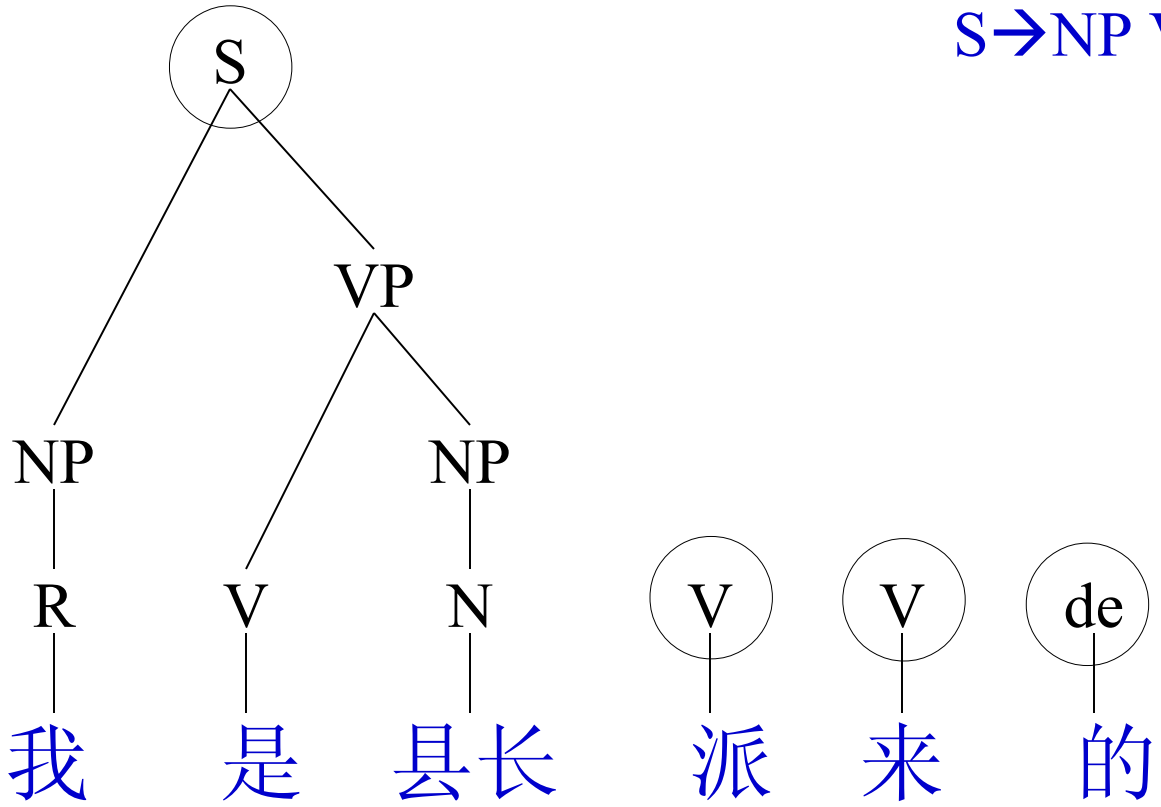
VP→V NP



自底向上分析法—示例 (5)

使用规则：

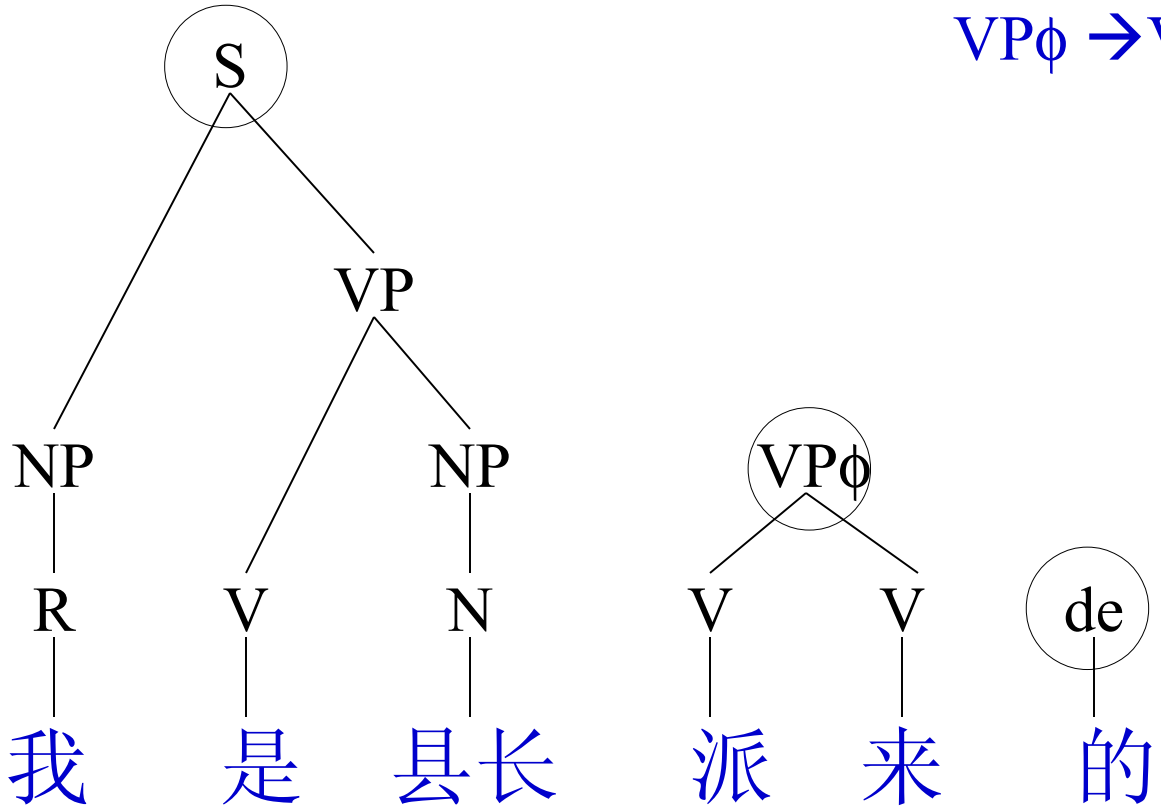
S→NP VP



自底向上分析法—示例 (6)

使用规则：

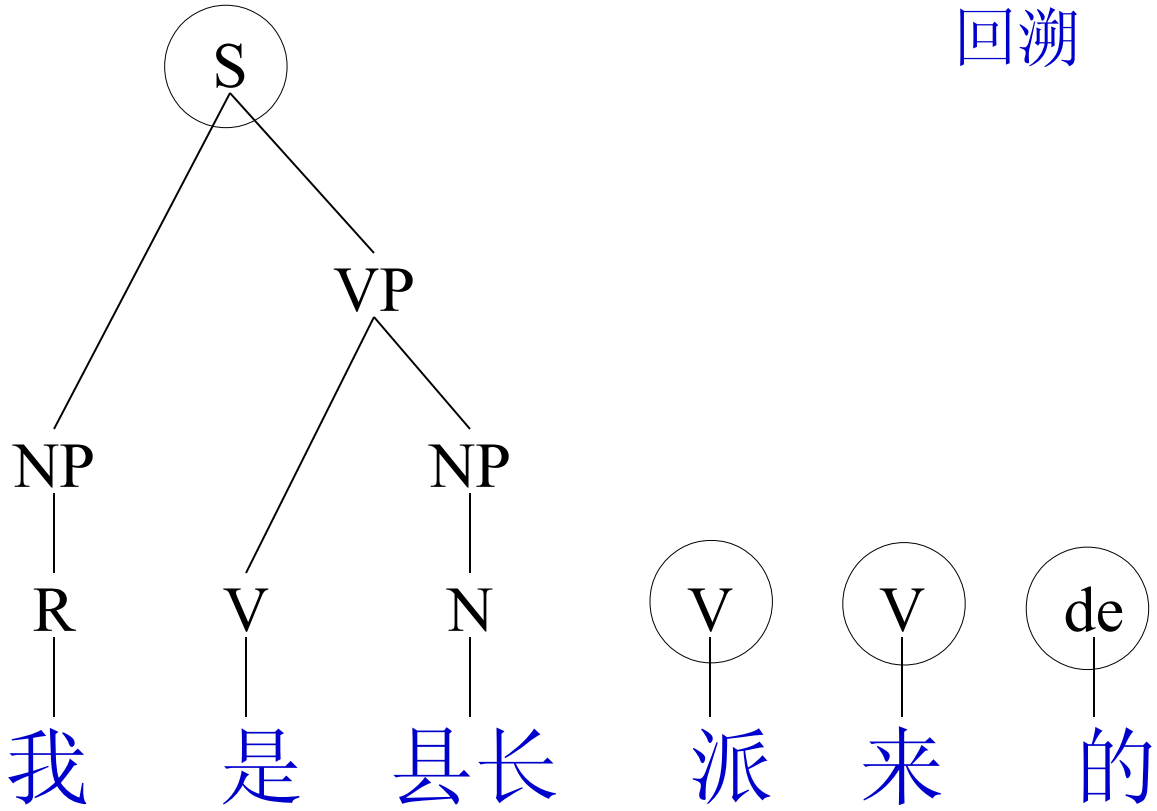
VPϕ →V V



自底向上分析法—示例 (7)

无规则可用

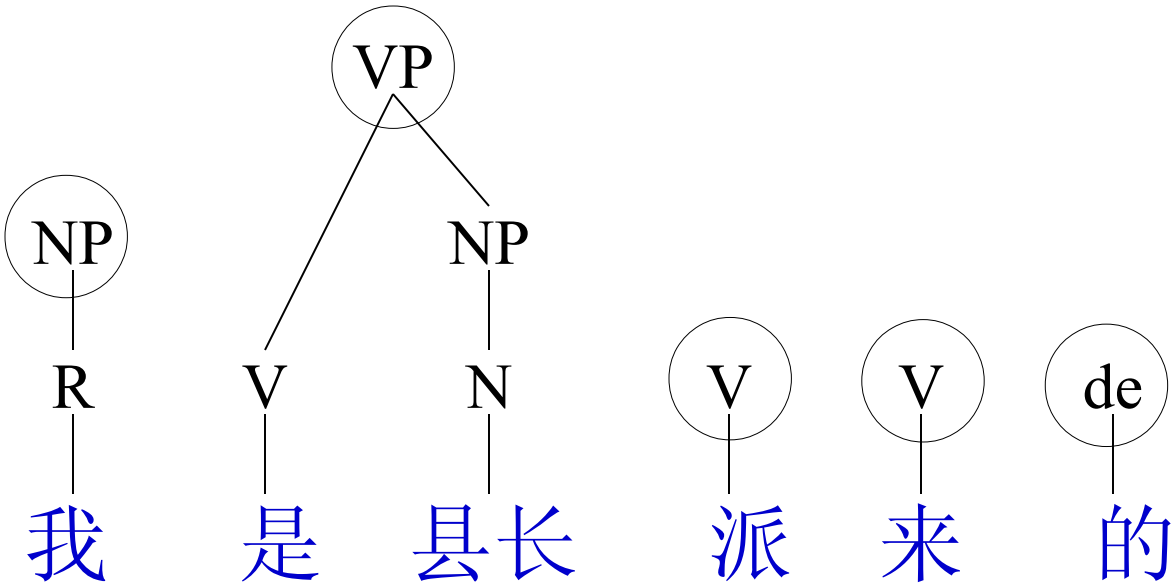
回溯



自底向上分析法—示例 (8)

无规则可用，

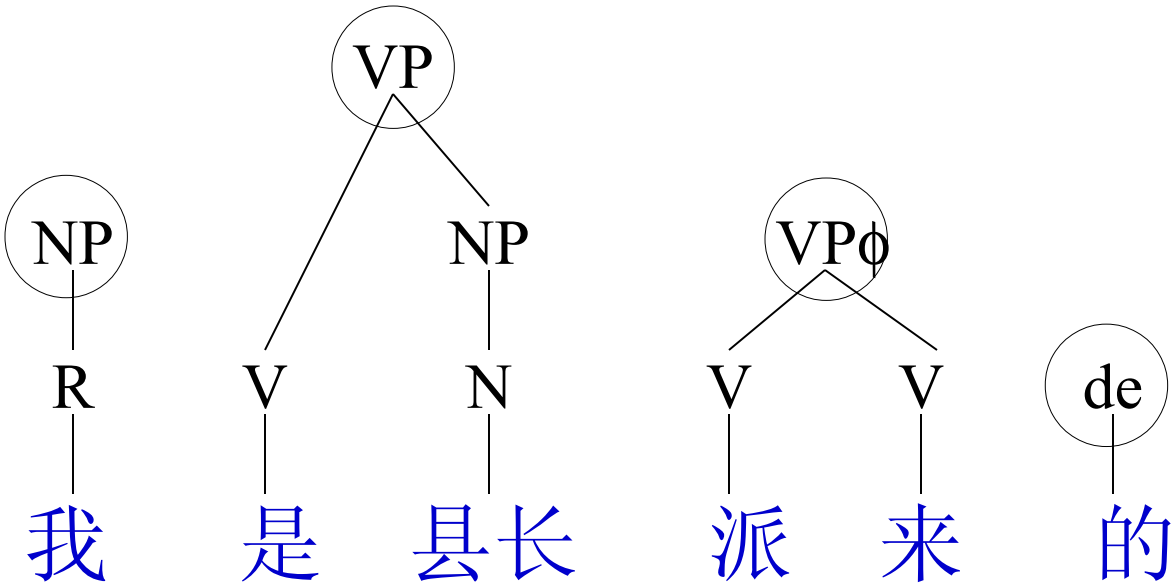
回溯



自底向上分析法—示例 (9)

使用规则：

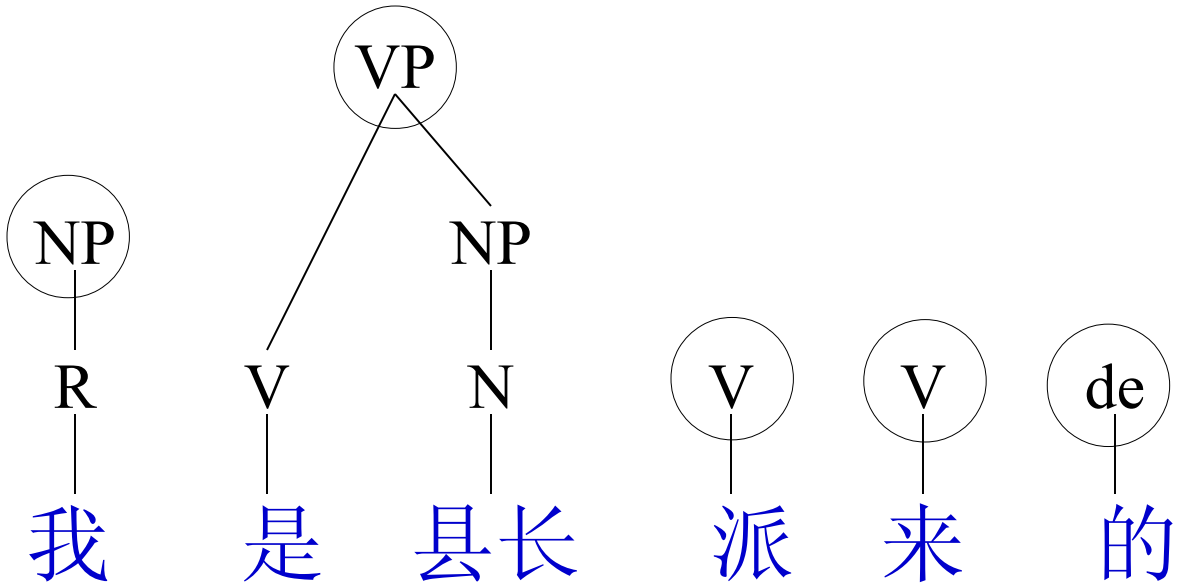
VPϕ →V V



自底向上分析法—示例 (10)

无规则可用，

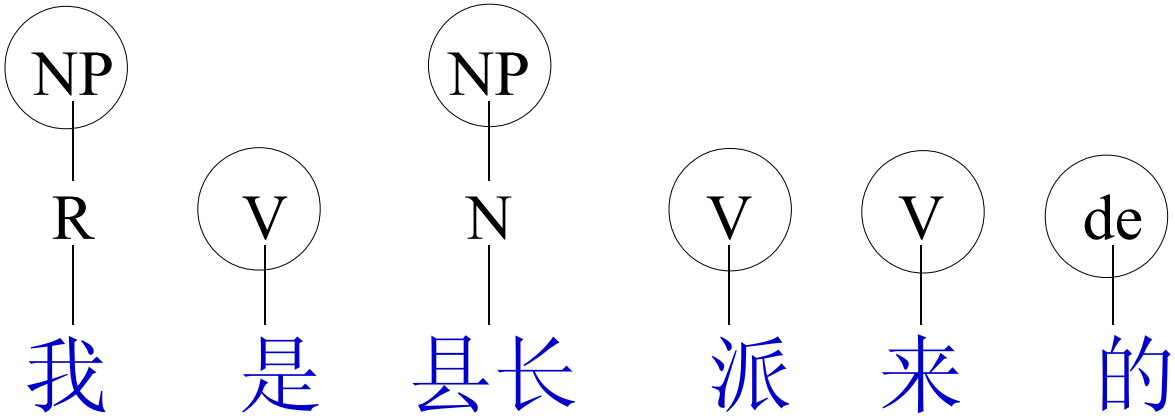
回溯



自底向上分析法—示例 (11)

无规则可用，

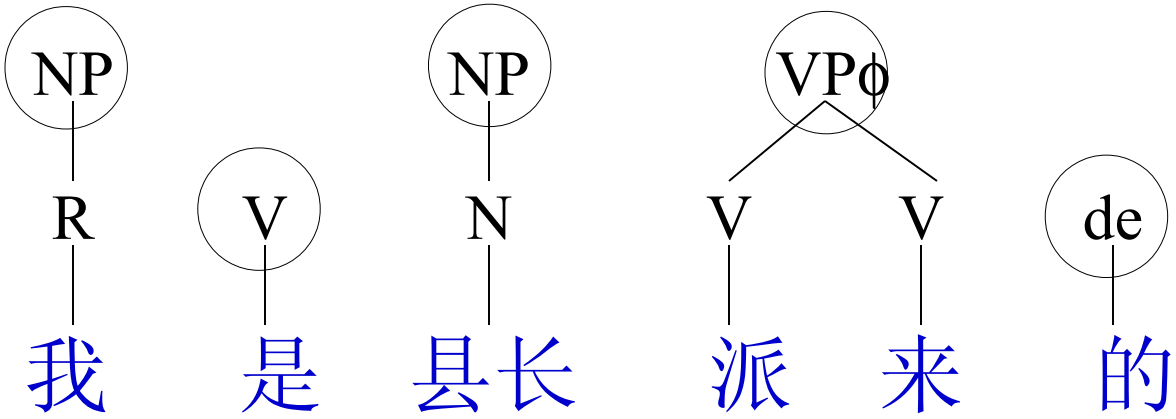
回溯



自底向上分析法—示例 (12)

使用规则：

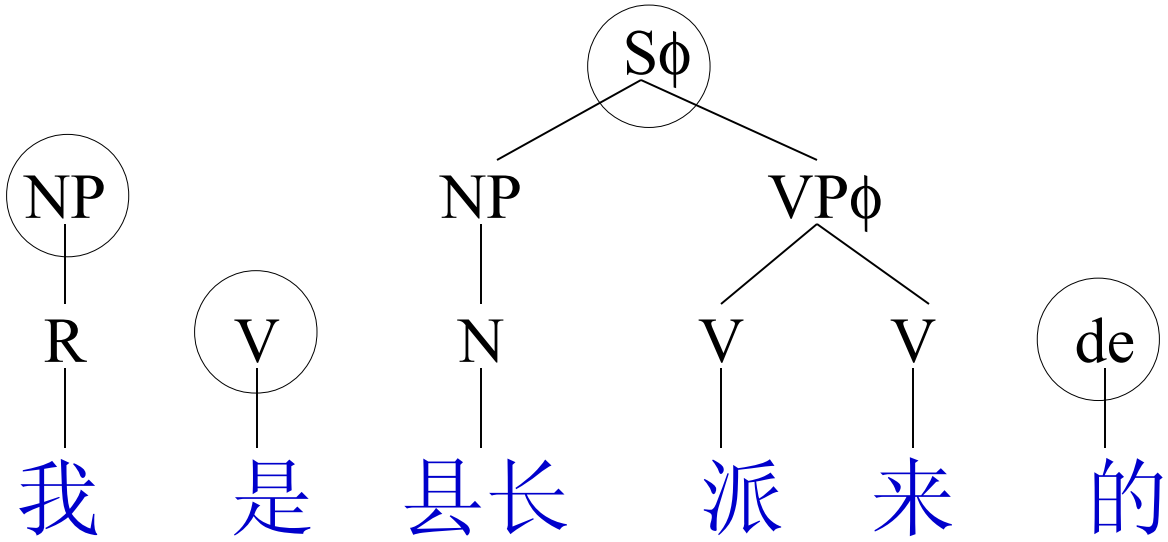
VPϕ →V V



自底向上分析法—示例 (13)

使用规则：

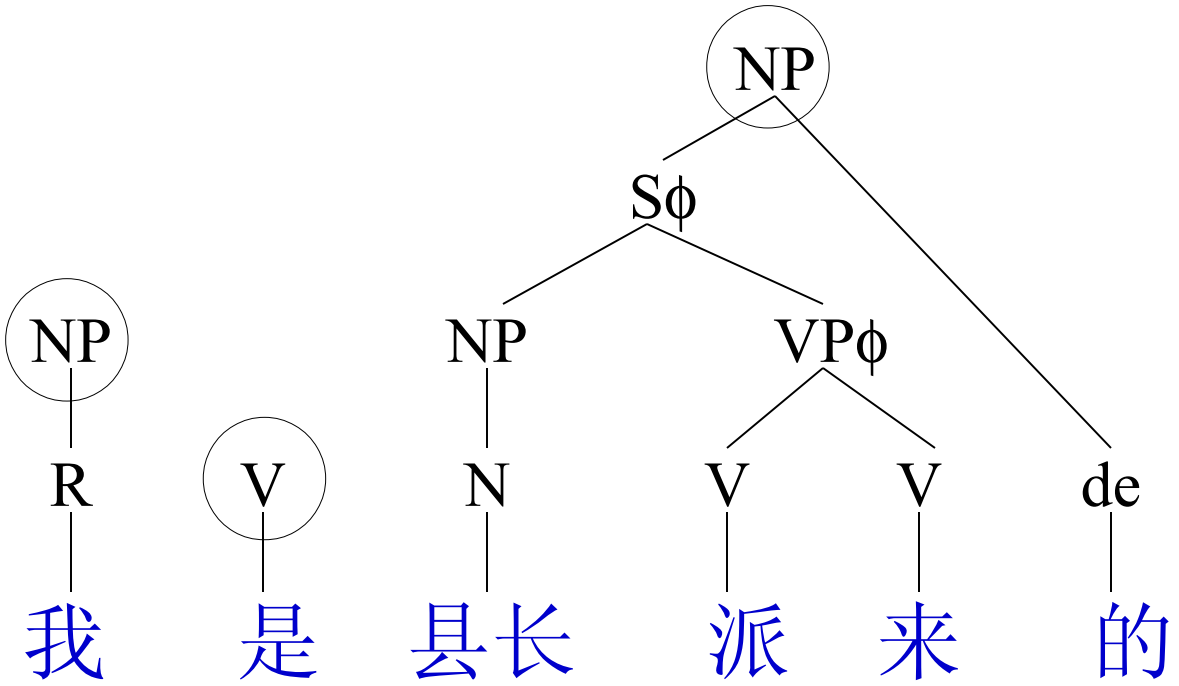
Sϕ →NP VPϕ



自底向上分析法—示例 (14)

使用规则：

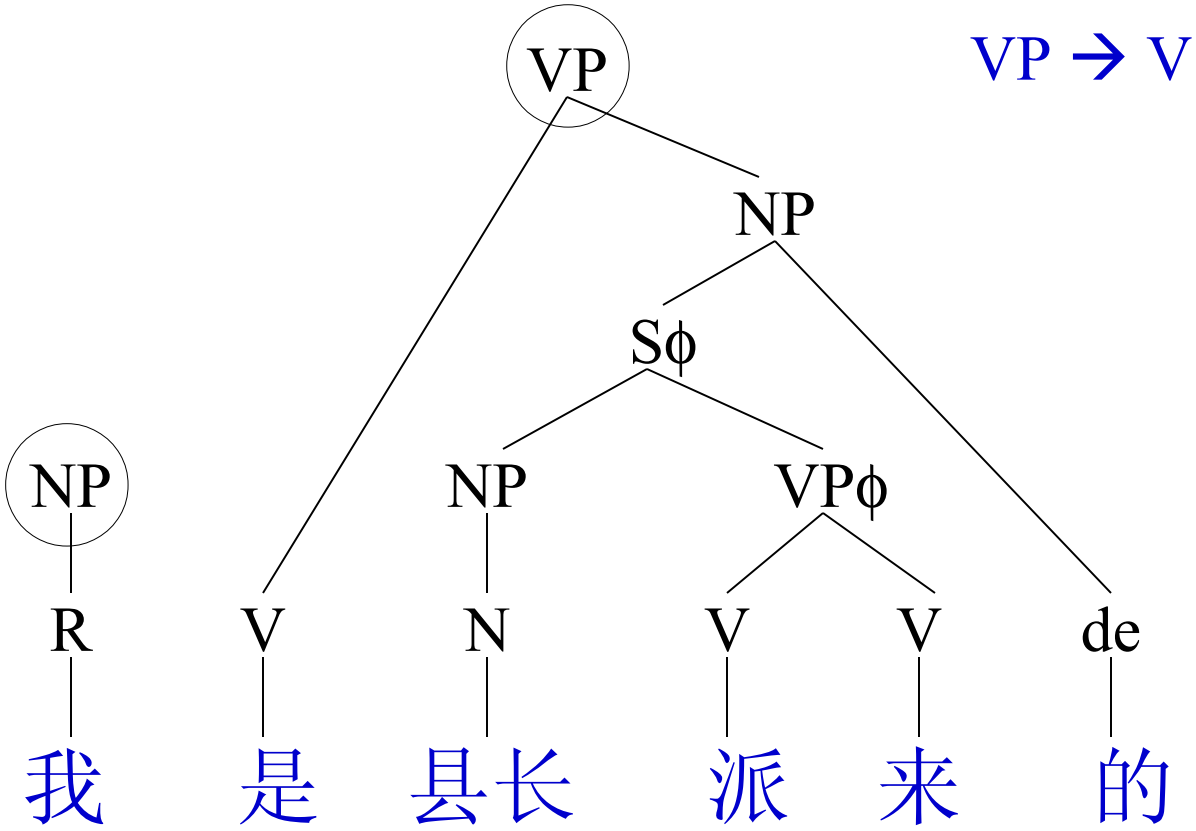
NP → Sϕ de



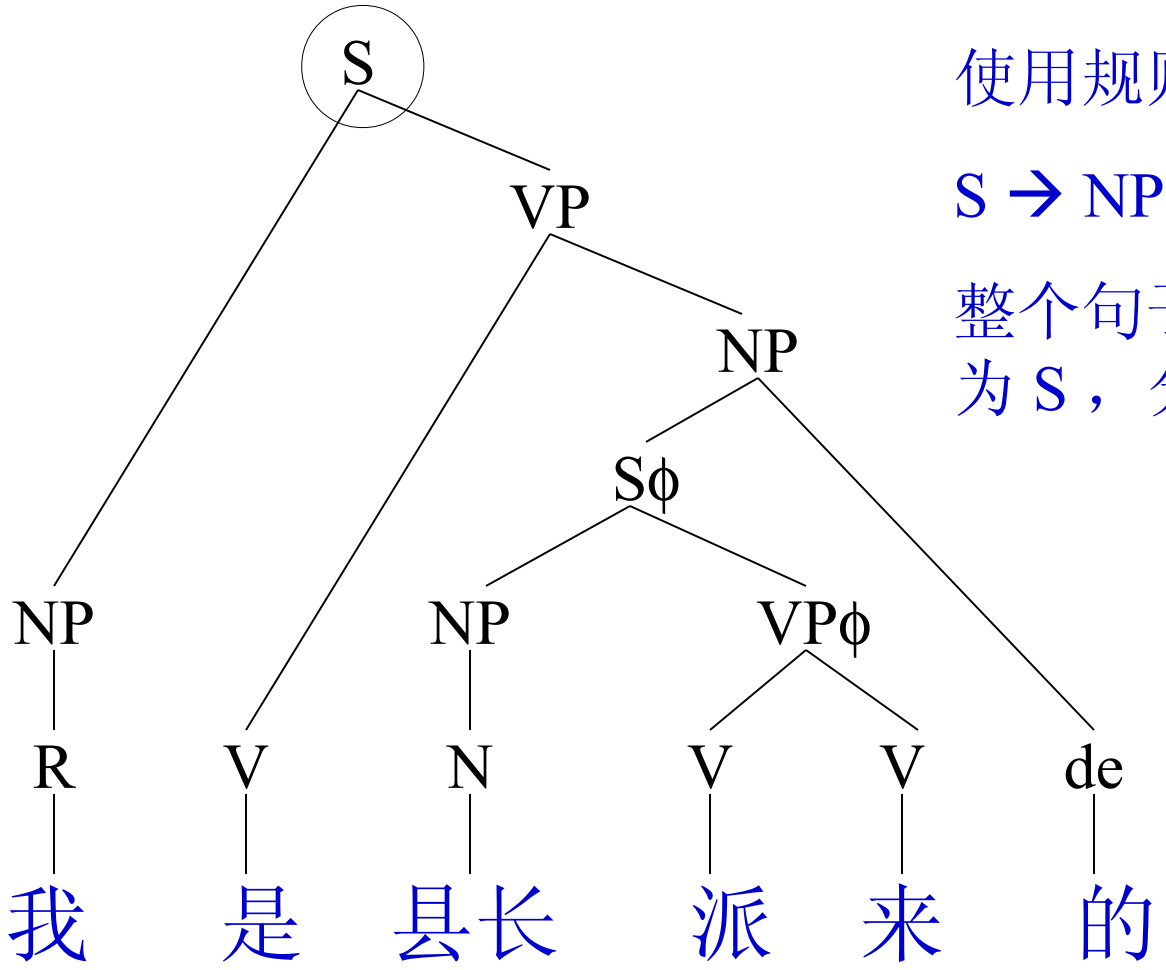
自底向上分析法—示例 (15)

使用规则：

VP → V NP



自底向上分析法—示例 (16)



使用规则：

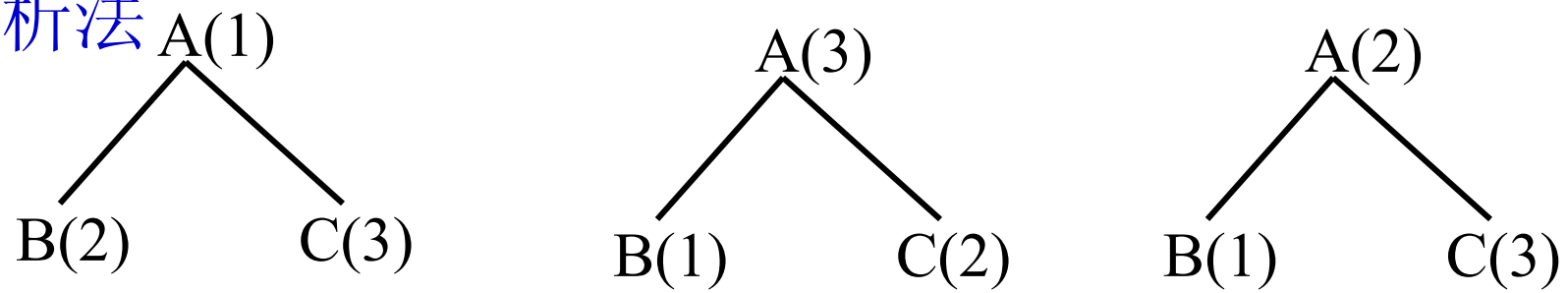
S → NP VP

整个句子归结为 S，分析成功

左角分析法—概述

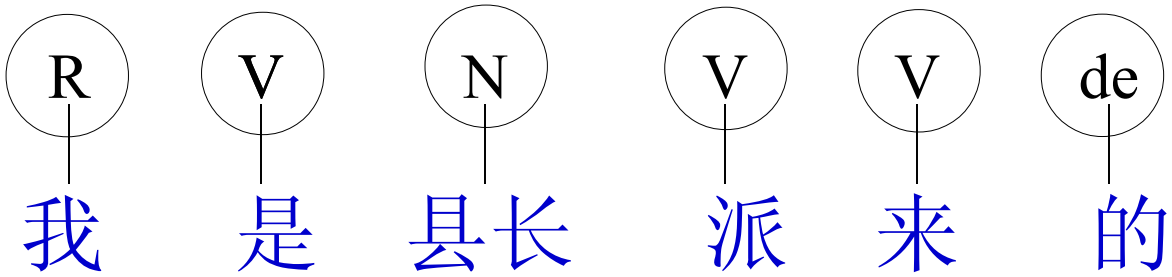
- 左角分析法是一种自顶向下和自底向上相结合的方法
- 所谓“左角 (Left Corner)” 是指任何一个句法子树中左下角的那个符号
- 比较：

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)

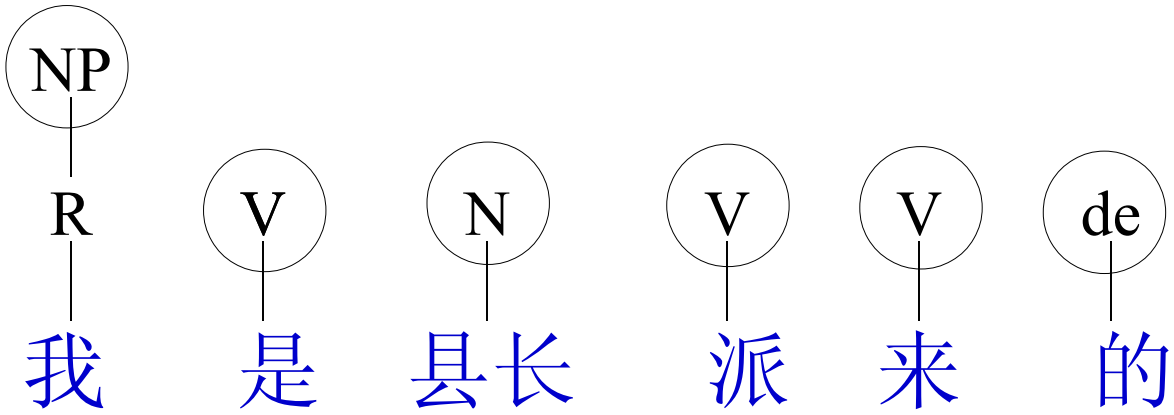
查词典



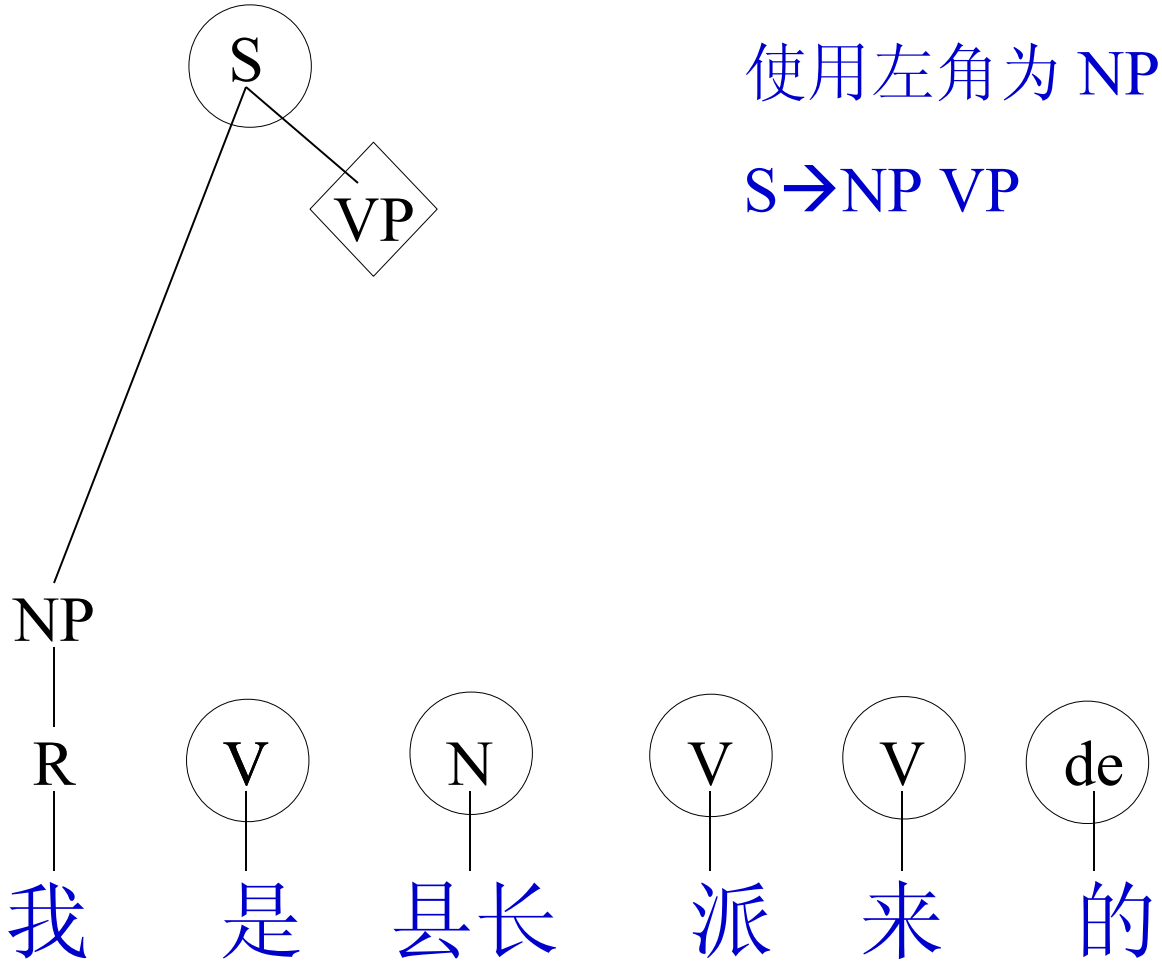
左角分析法—示例 (2)

使用左角为 R 的规则：

NP→R



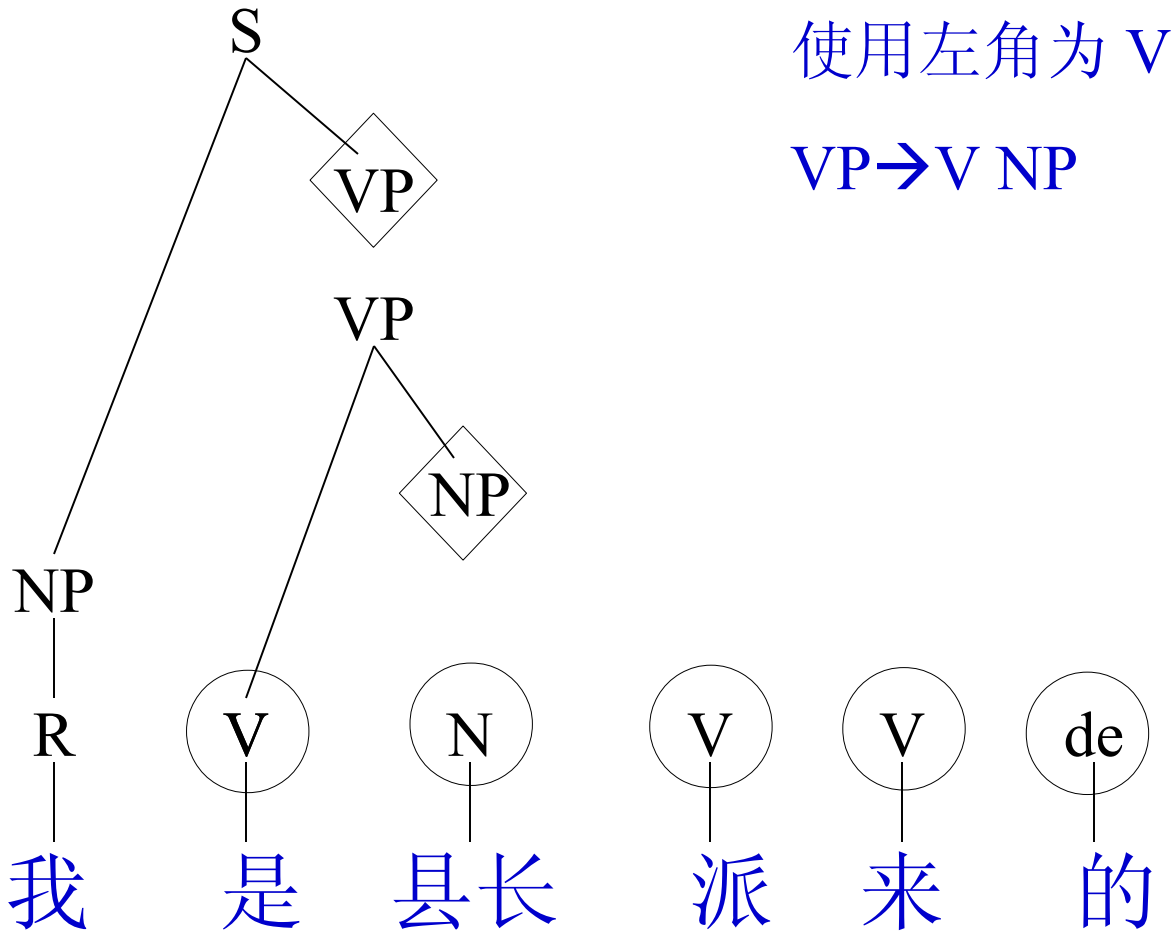
左角分析法—示例 (3)



使用左角为 NP 的规则：

S→NP VP

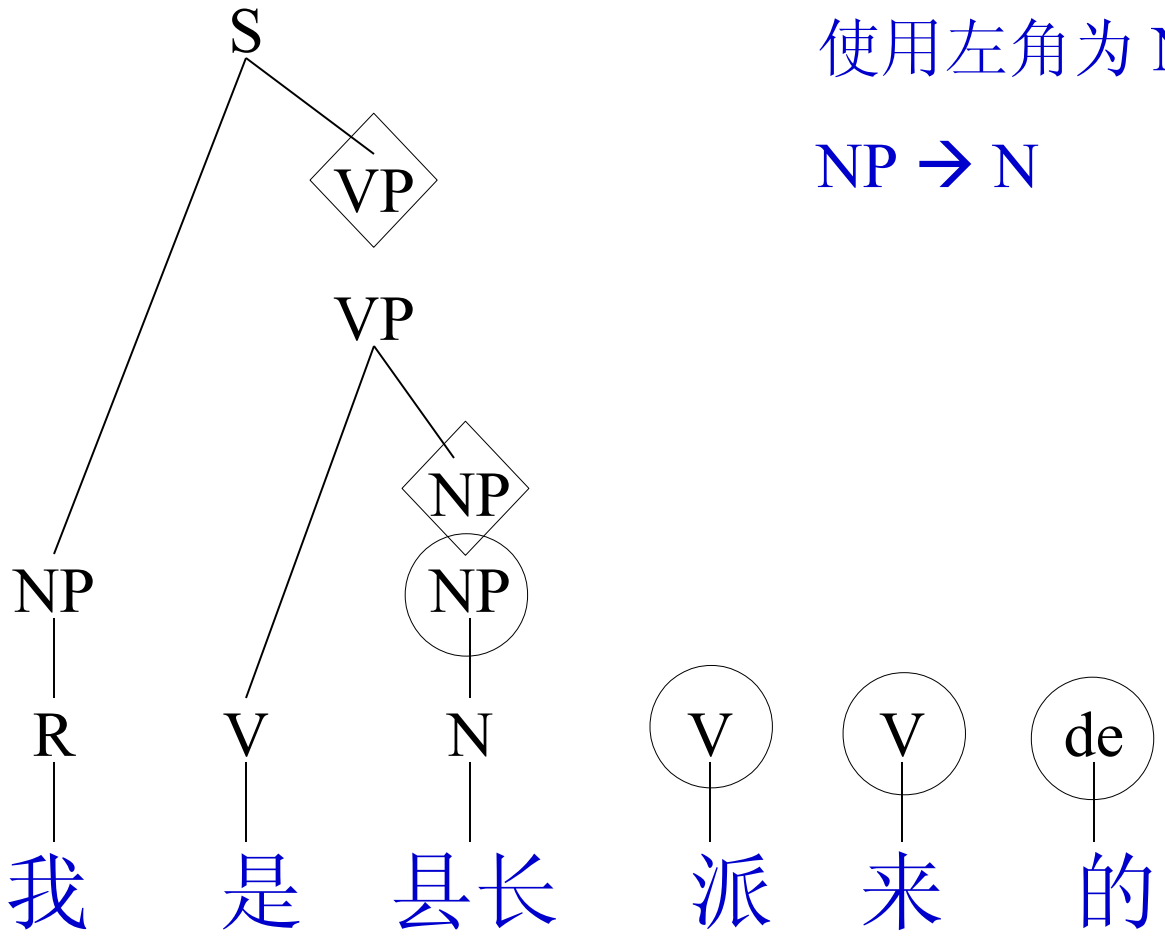
左角分析法—示例 (4)



使用左角为 V 的规则：

VP→V NP

左角分析法—示例 (5)

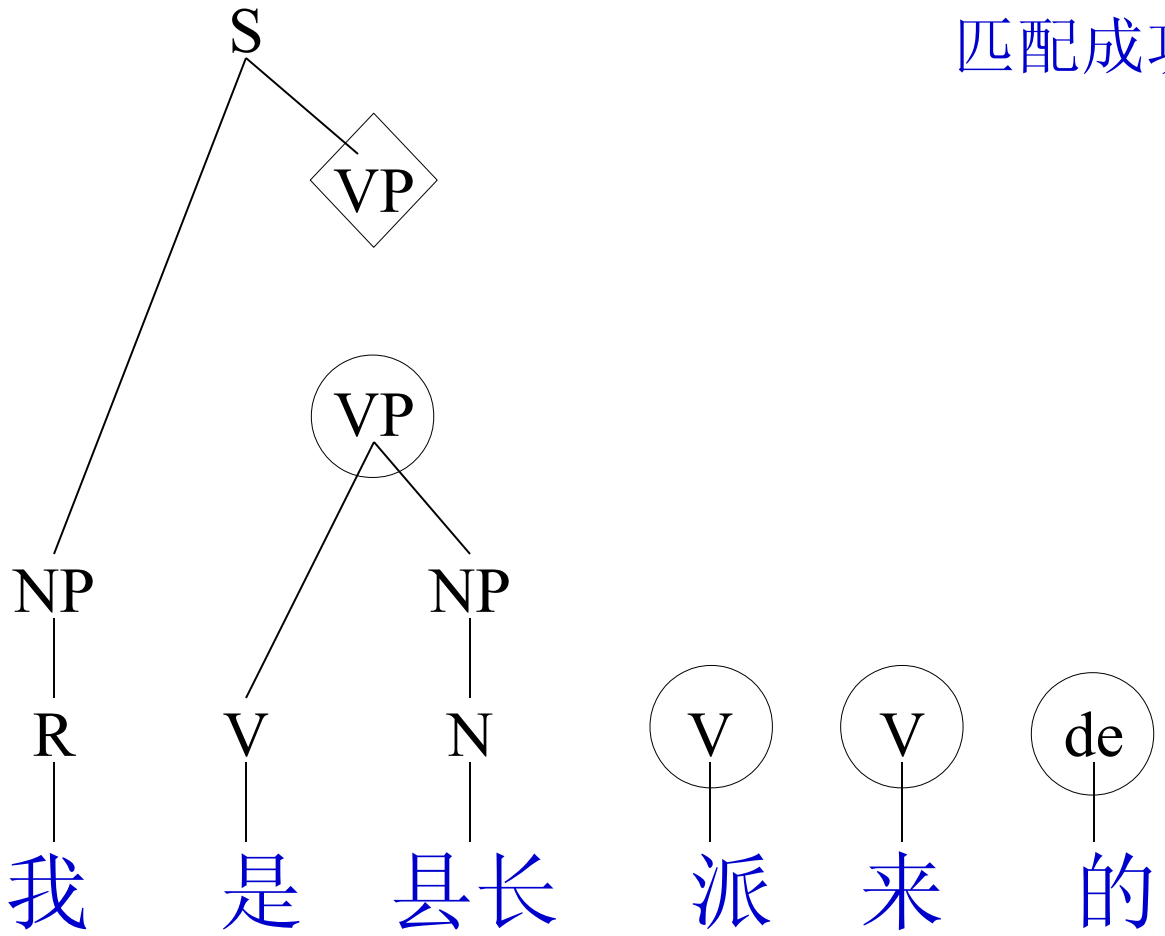


使用左角为 N 规则：

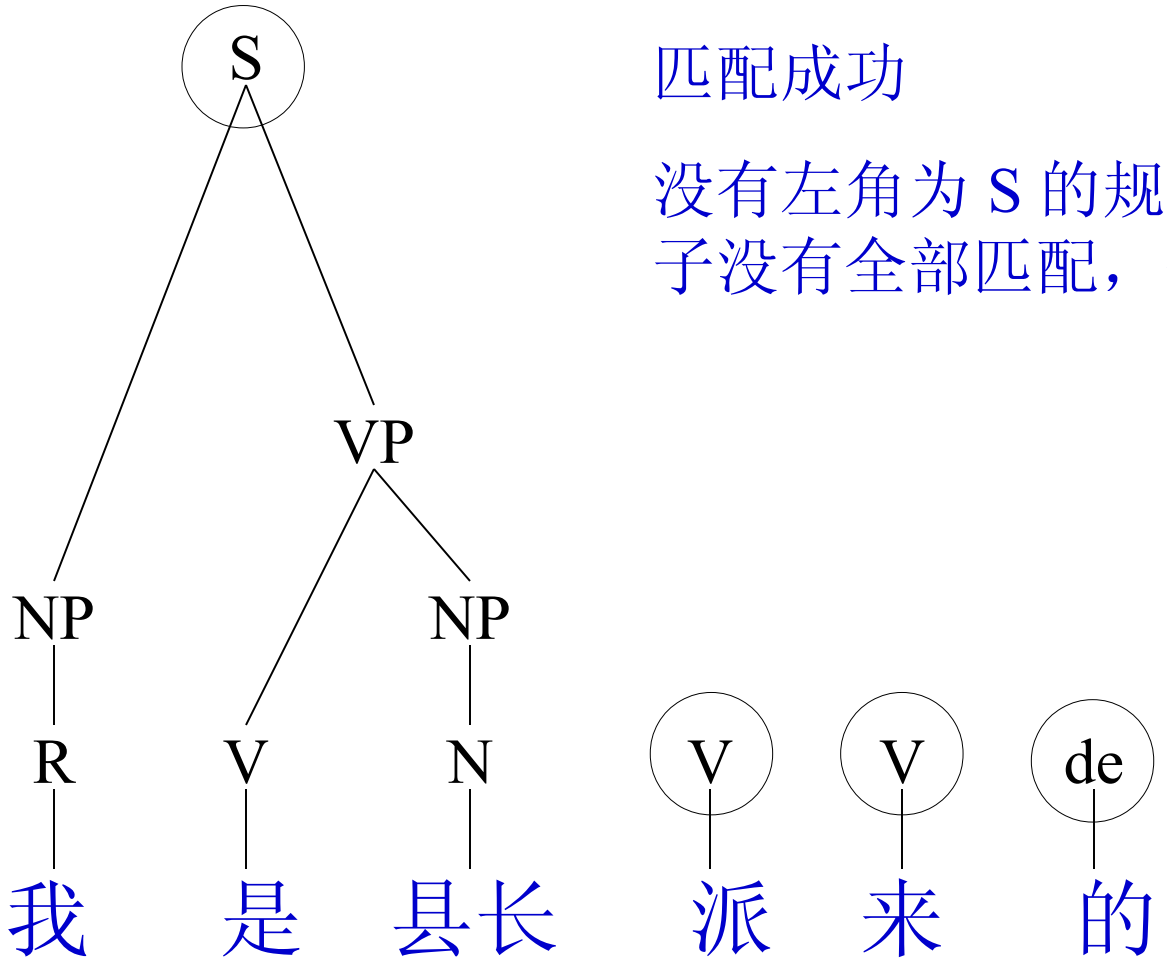
NP → N

左角分析法—示例 (6)

匹配成功



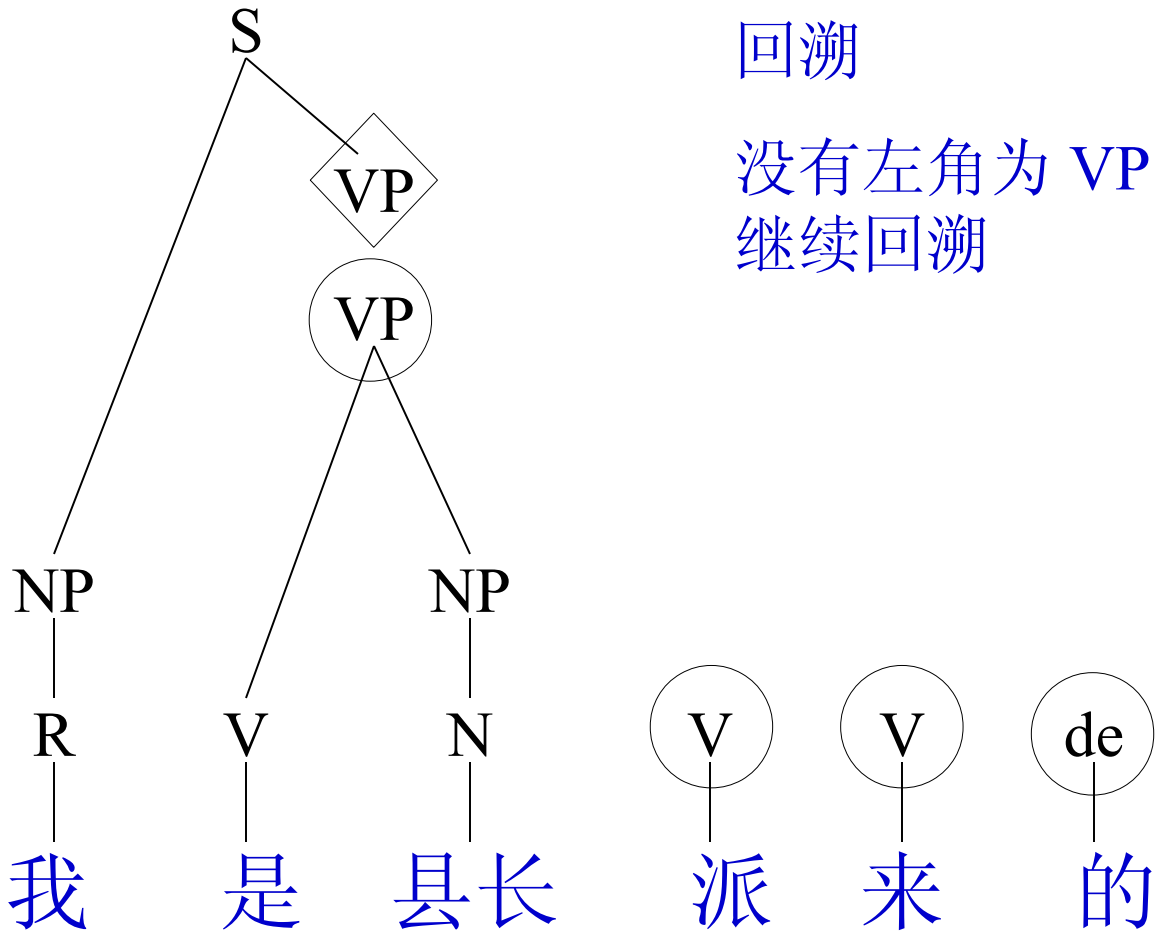
左角分析法—示例 (7)



匹配成功

没有左角为 S 的规则，且句子没有全部匹配，分析失败

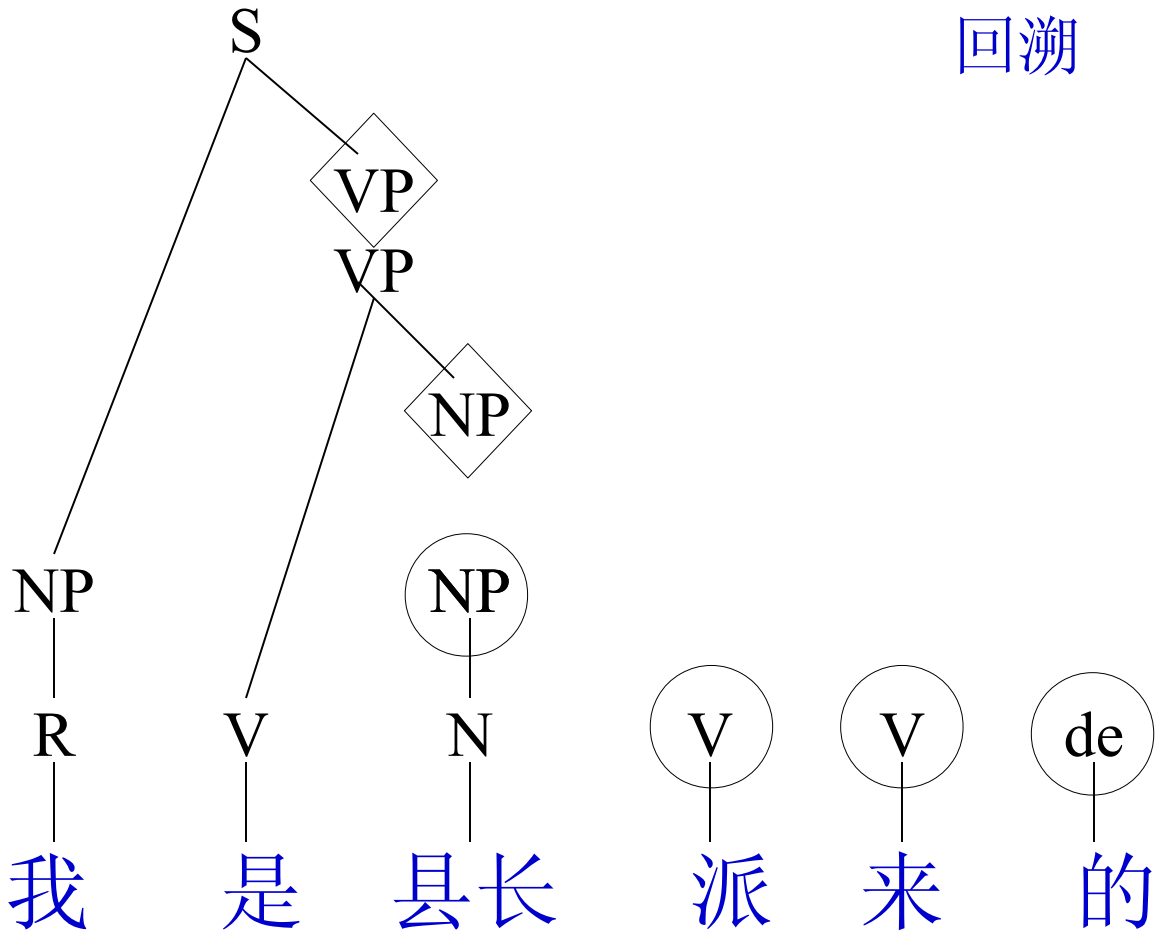
左角分析法—示例 (8)



回溯

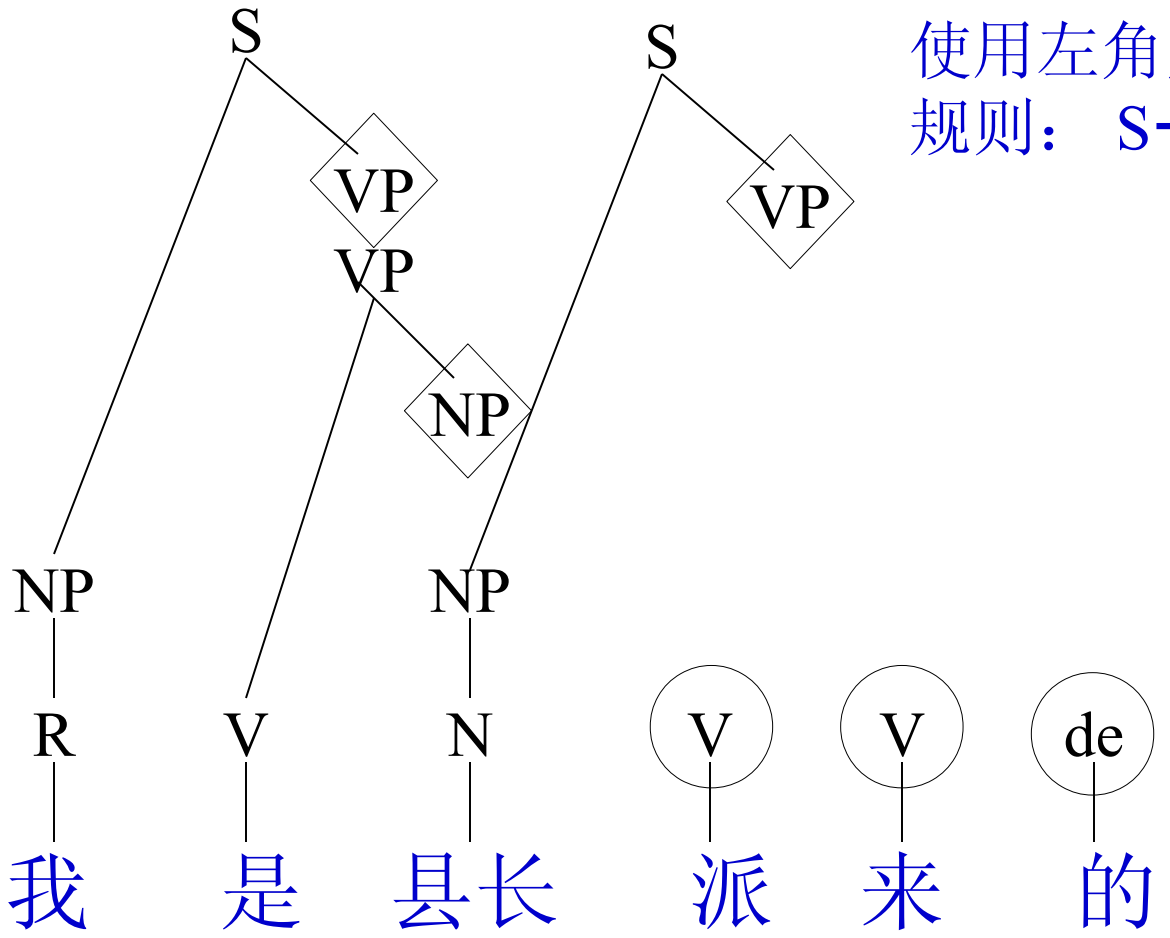
没有左角为 VP 的规则，继续回溯

左角分析法—示例 (9)



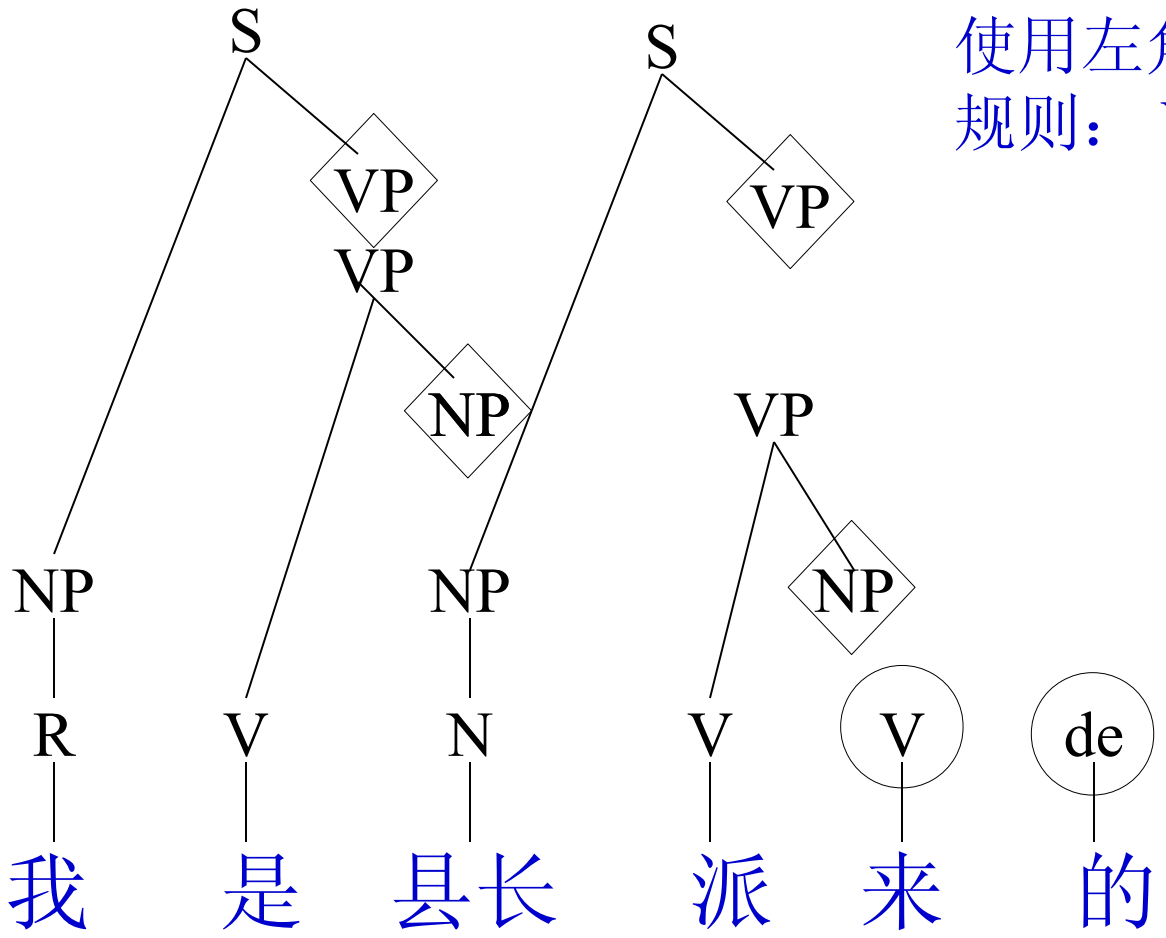
回溯

左角分析法—示例 (10)



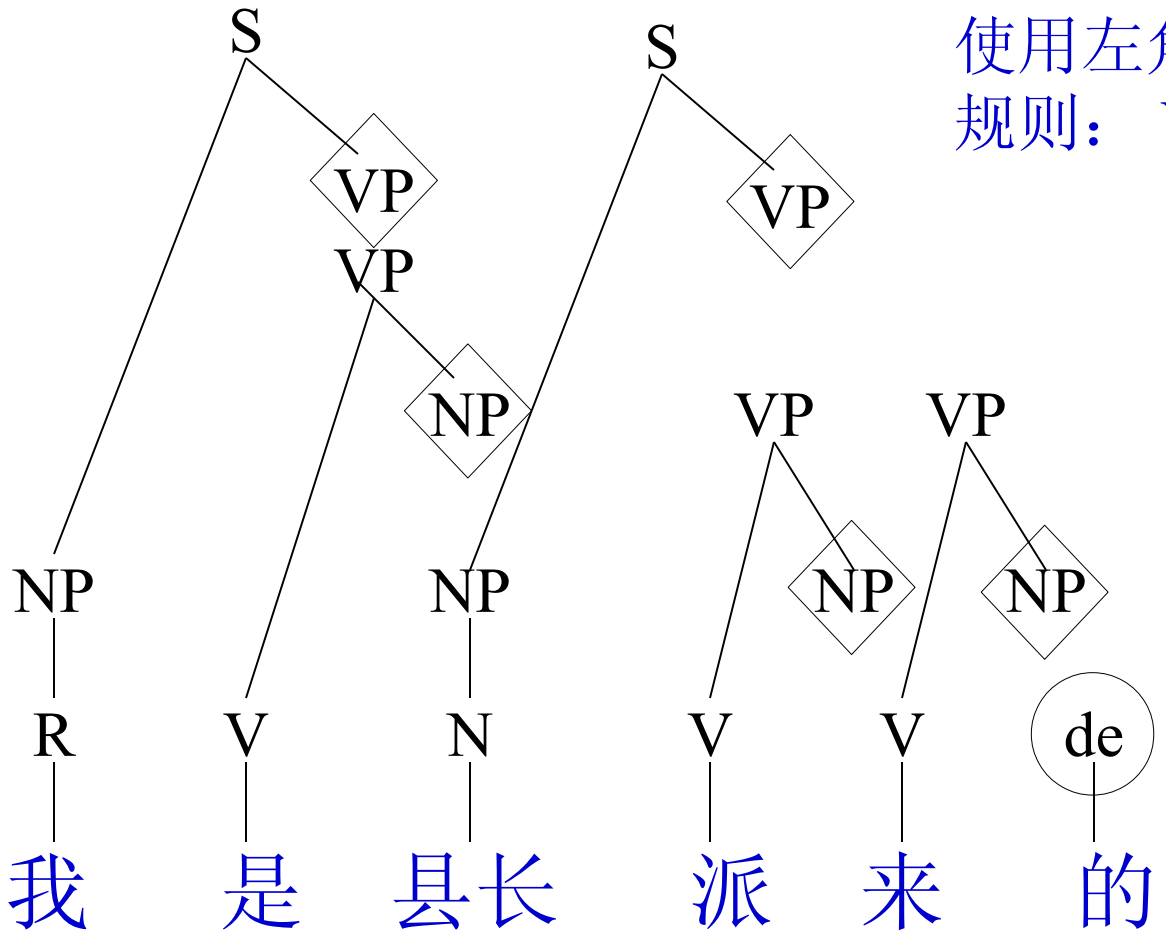
使用左角为 NP 的规则： S→NP VP

左角分析法—示例 (11)



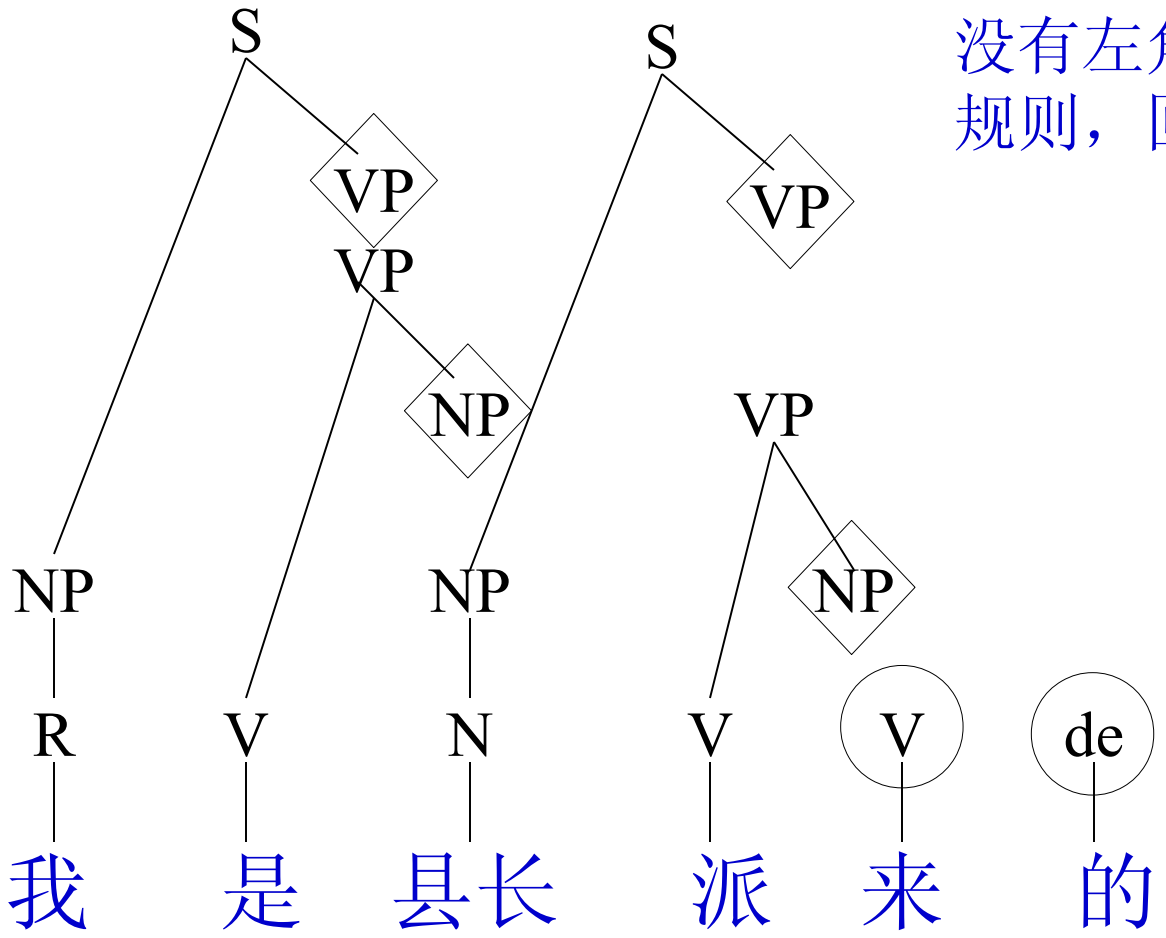
使用左角为 V 的规则： VP→V NP

左角分析法—示例 (12)



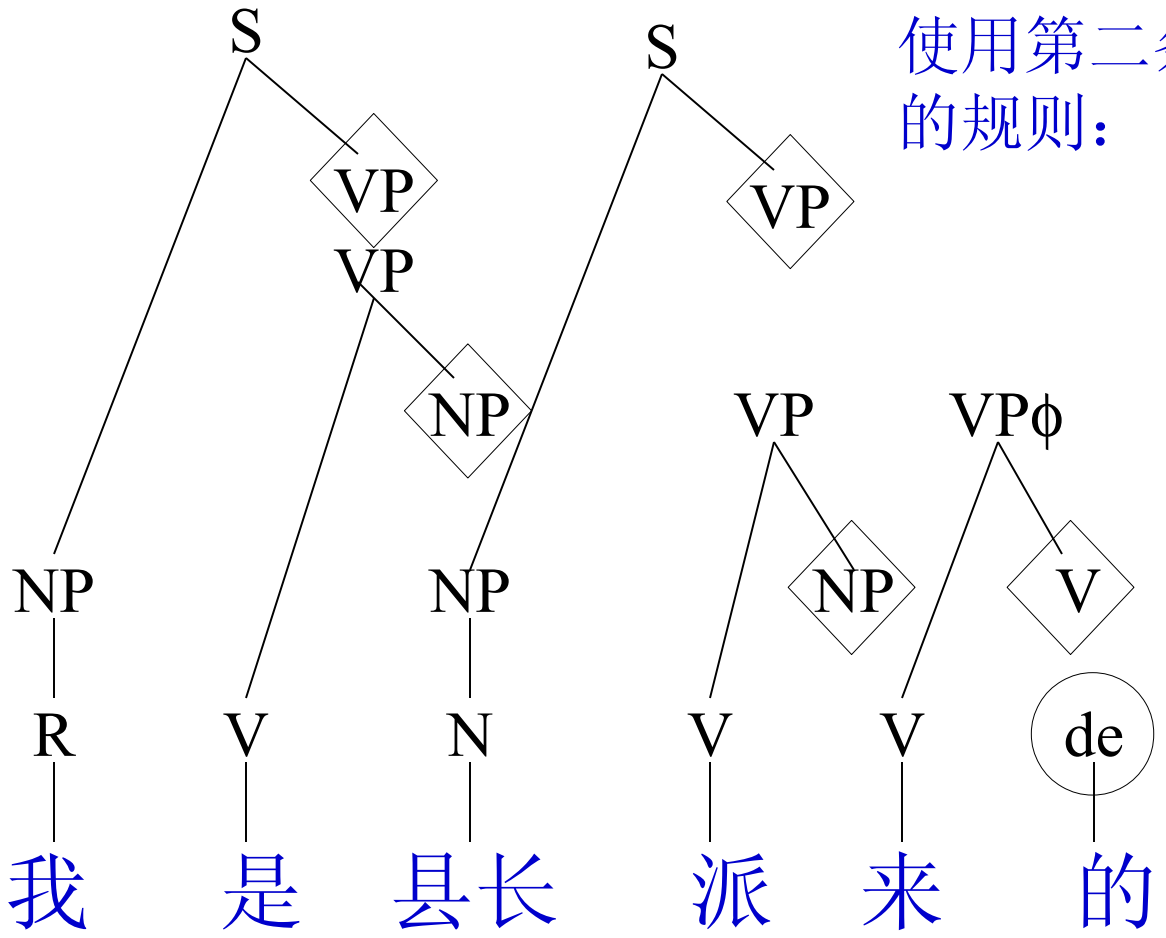
使用左角为 V 的规则： VP→V NP

左角分析法—示例 (13)



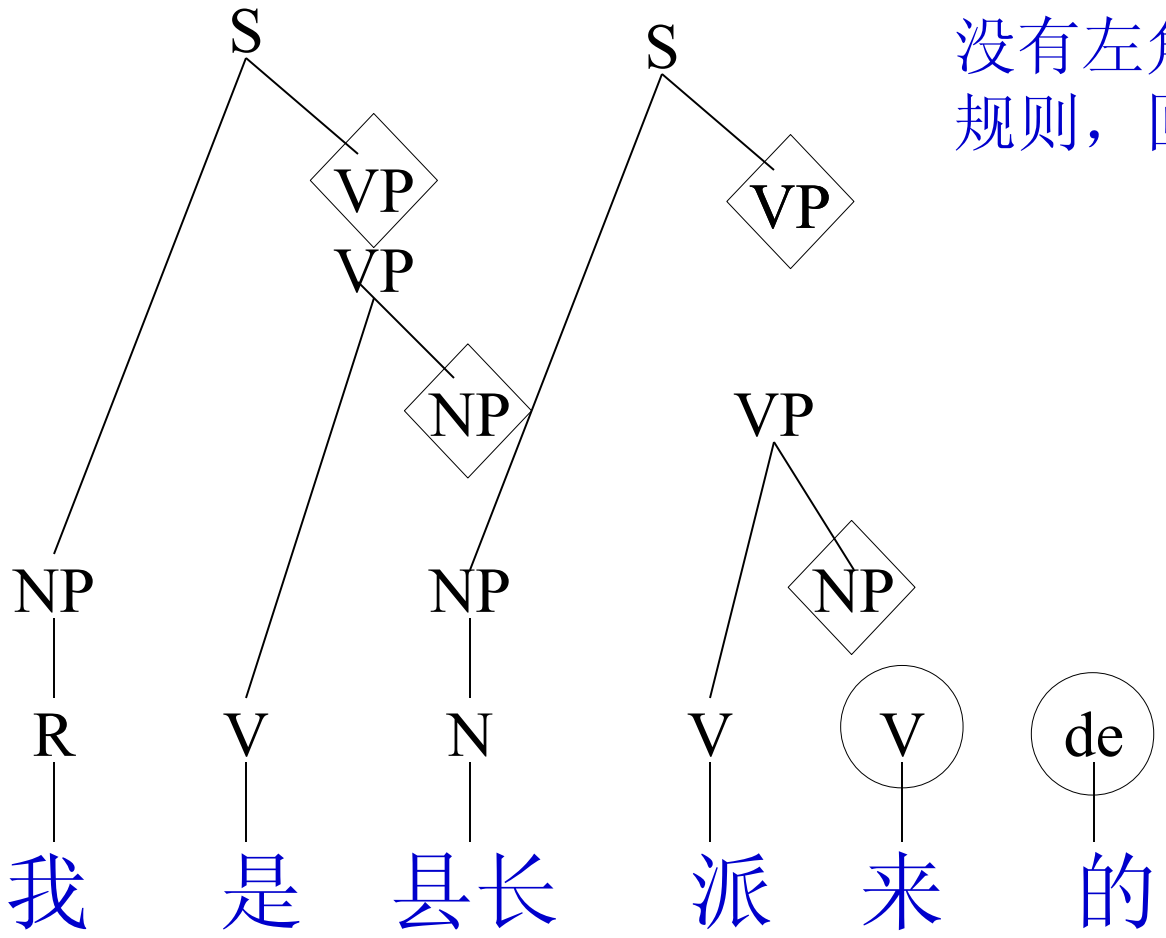
没有左角为 de 的规则，回溯

左角分析法—示例 (14)



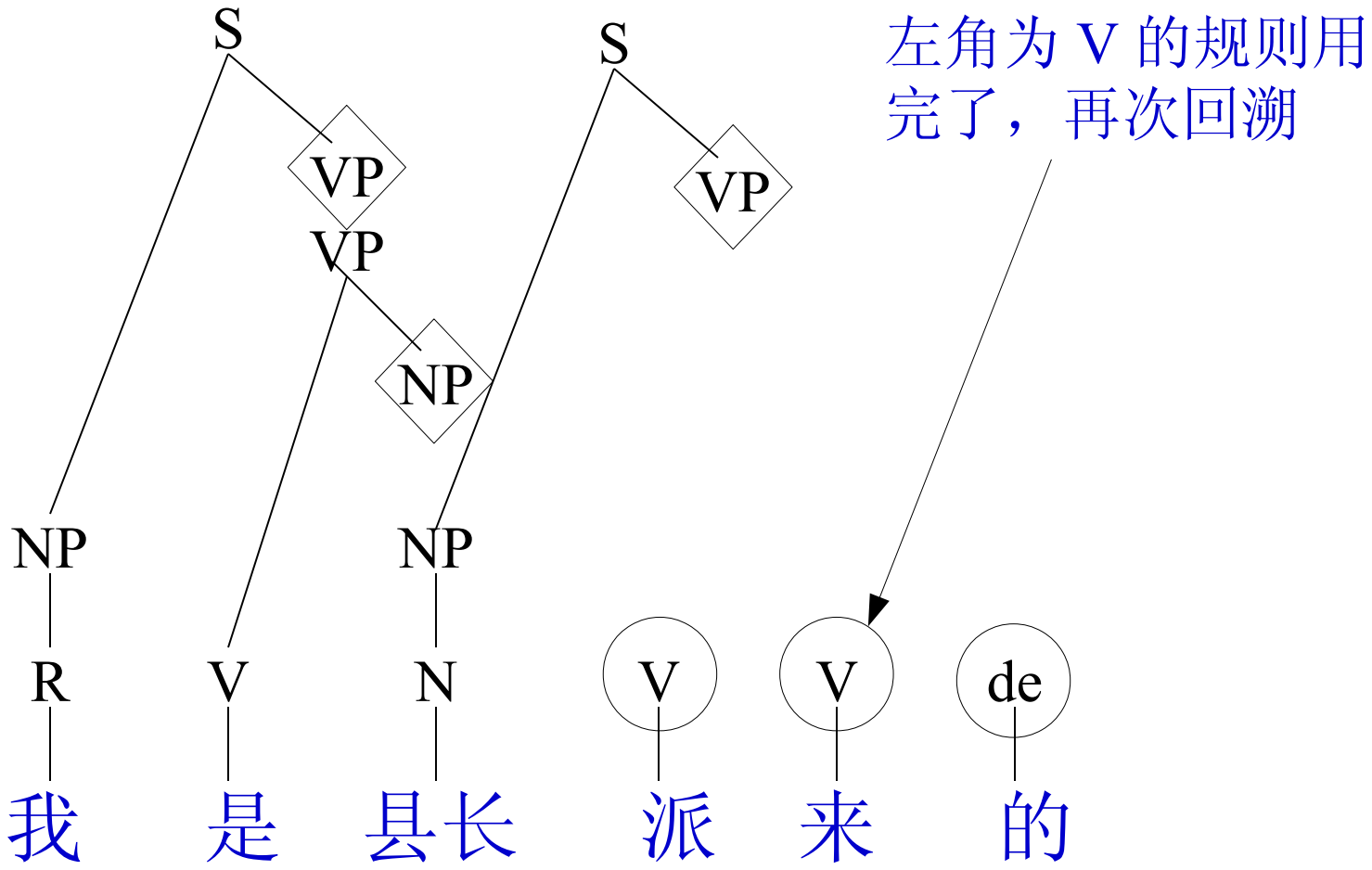
使用第二条左角为 V 的规则： VPϕ →V V

左角分析法—示例 (15)

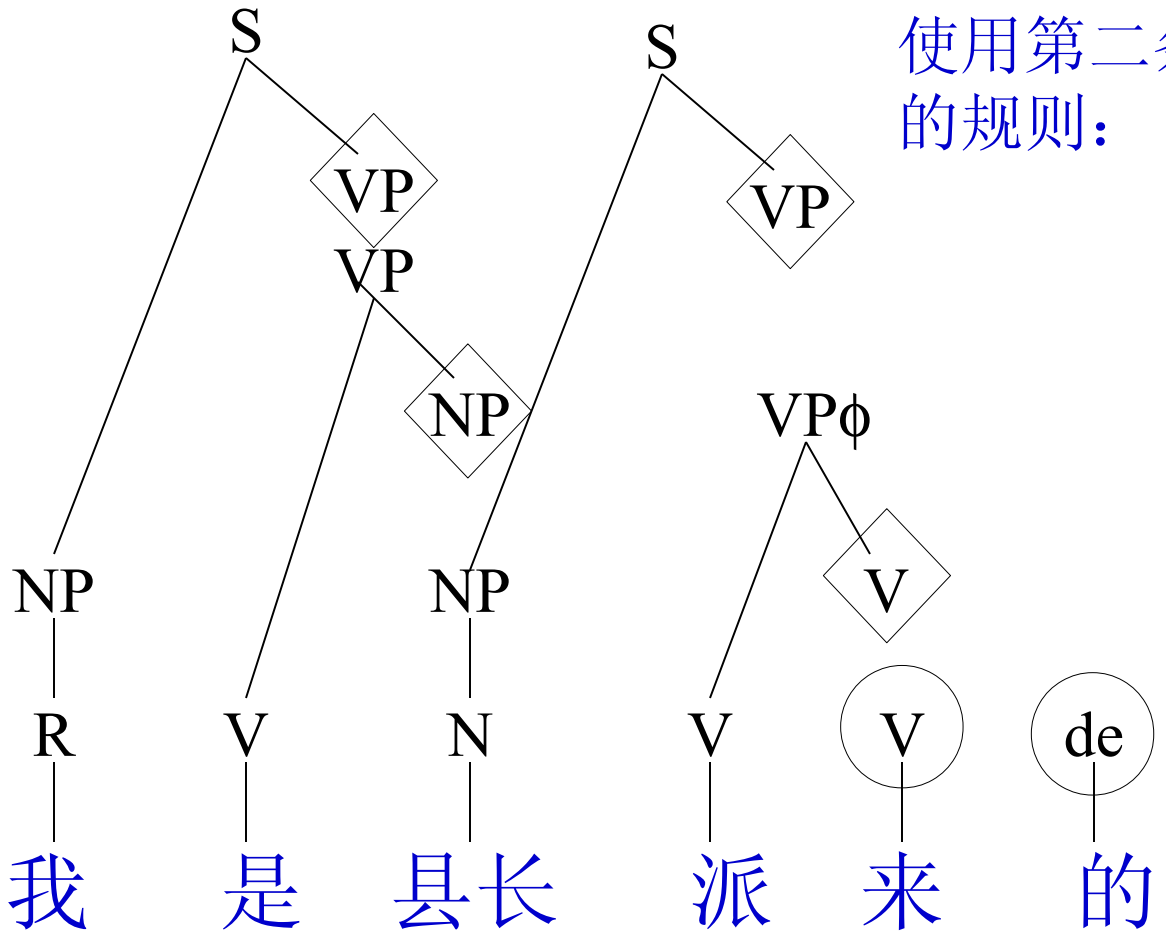


没有左角为 de 的规则，回溯

左角分析法—示例 (16)

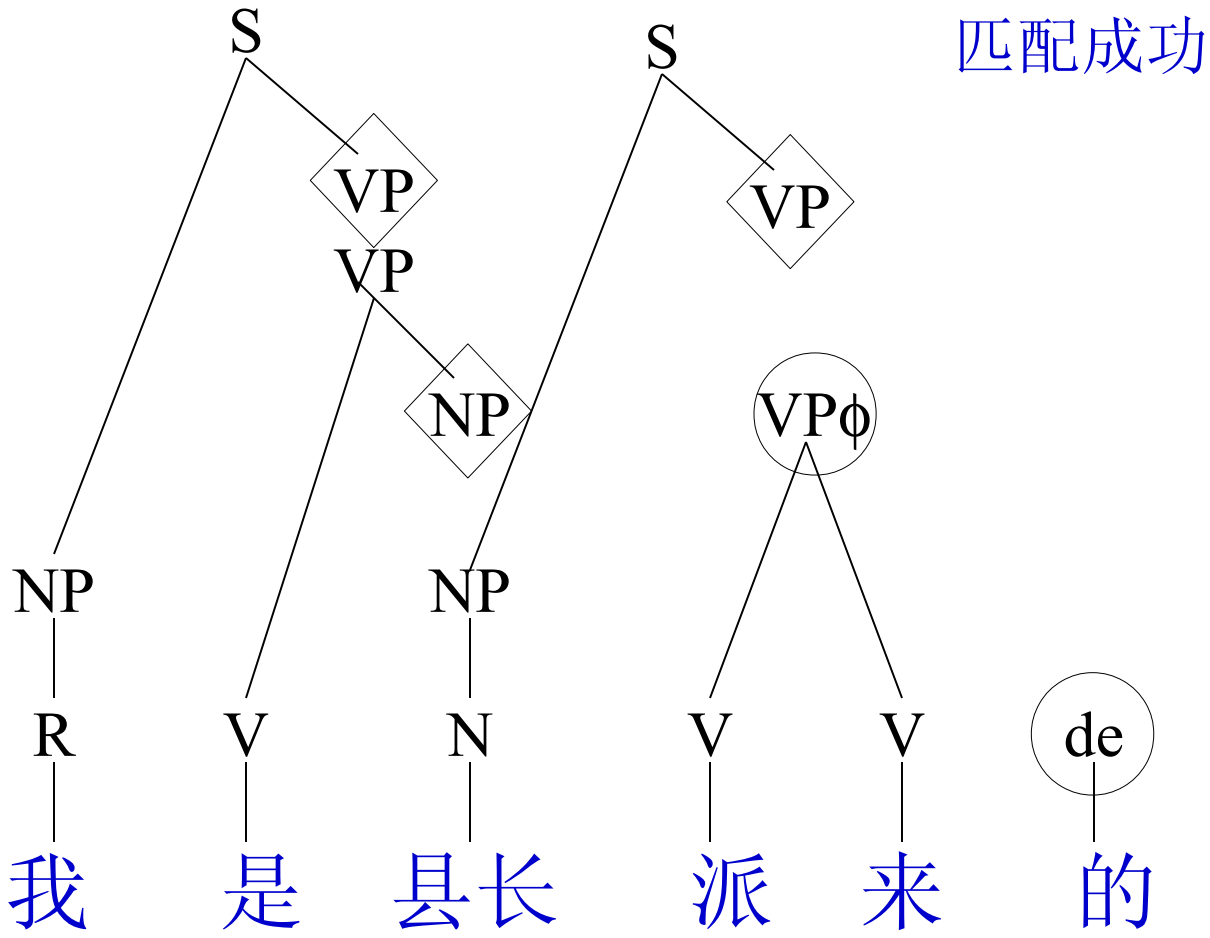


左角分析法—示例 (17)

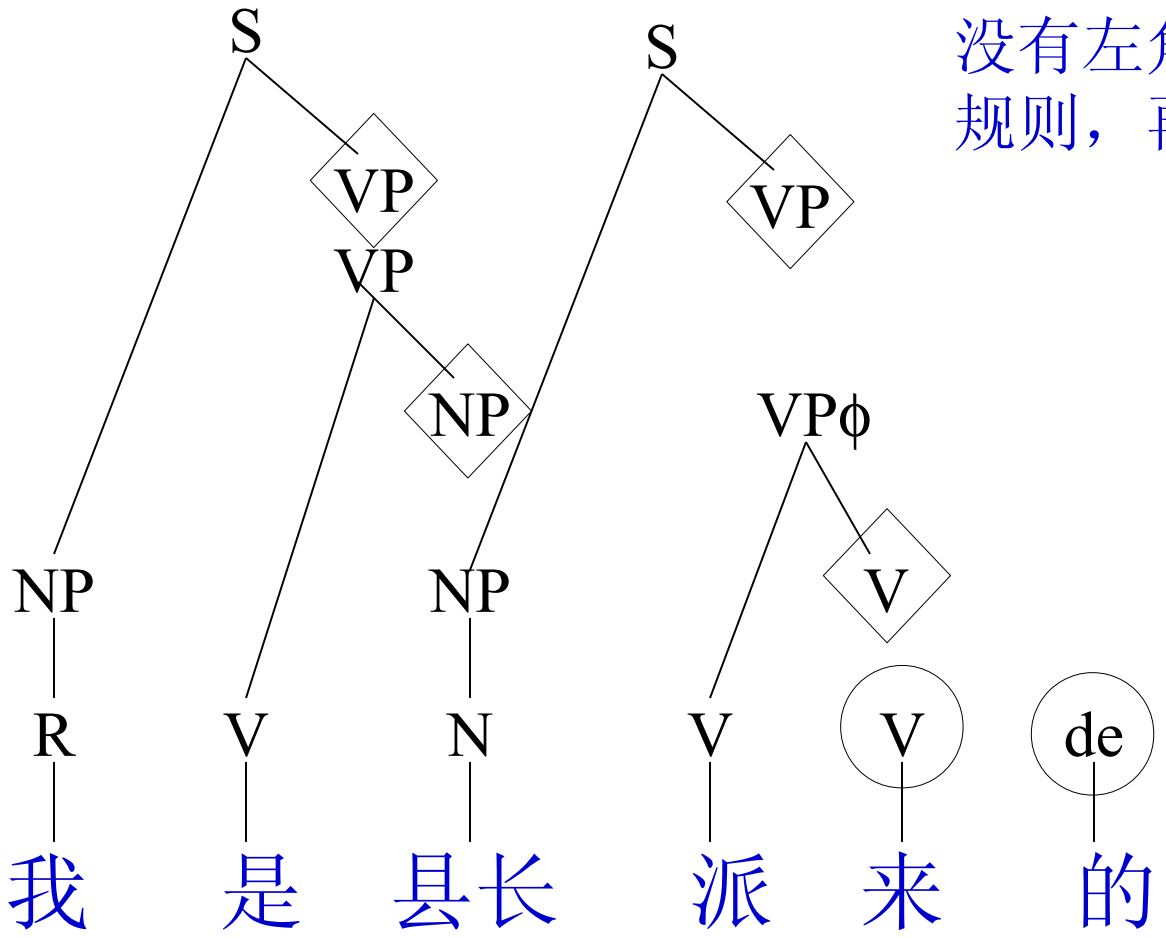


使用第二条左角为 V 的规则： VPϕ →V V

左角分析法—示例 (18)

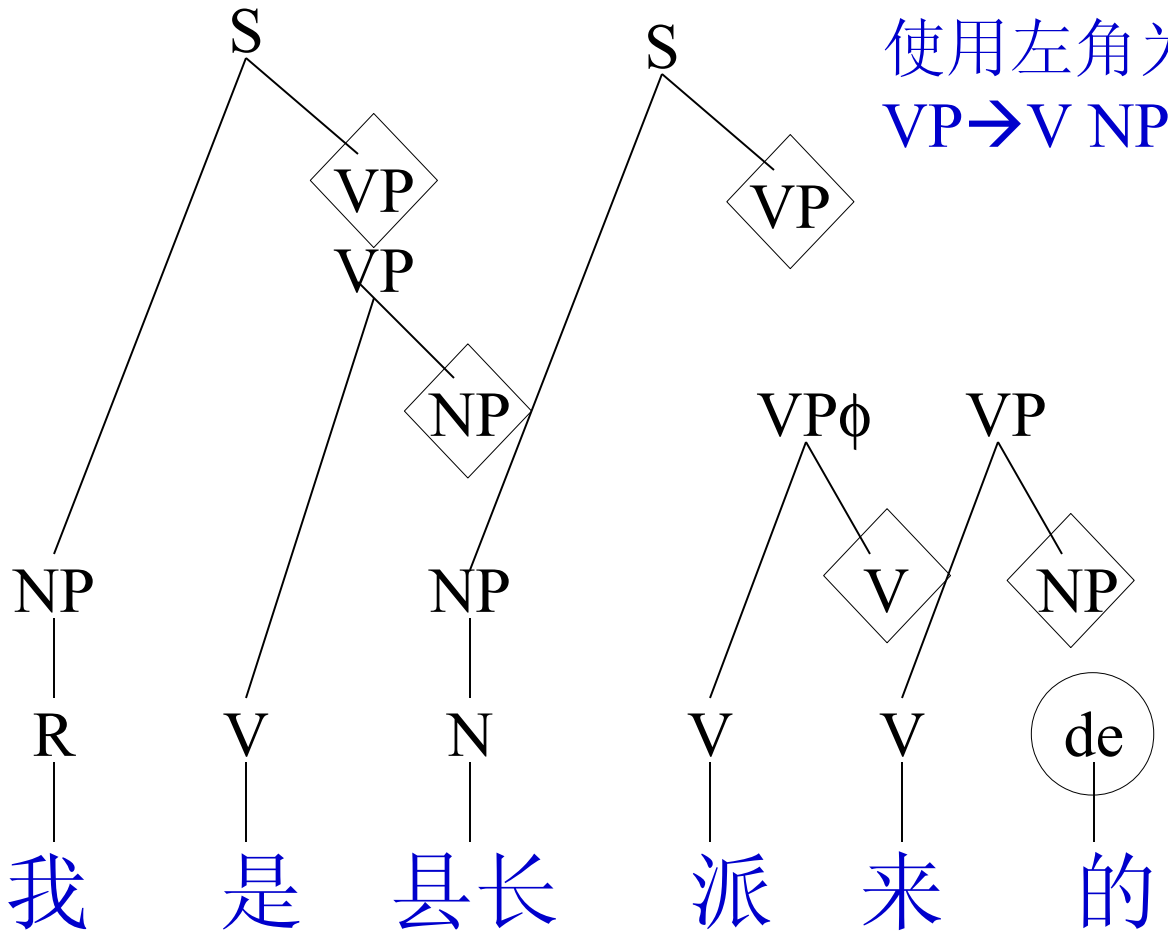


左角分析法—示例 (19)



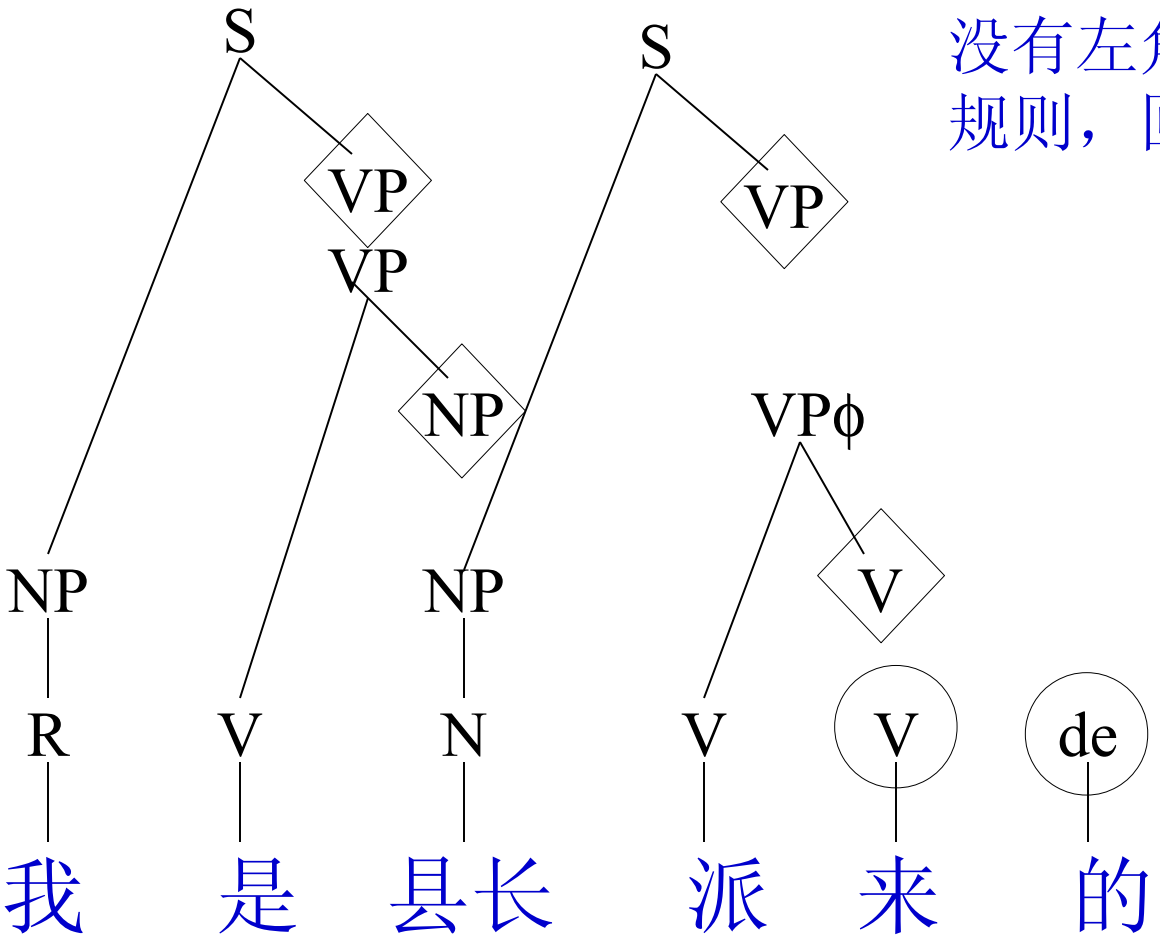
没有左角为 VPϕ 的规则，再次回溯

左角分析法—示例 (20)



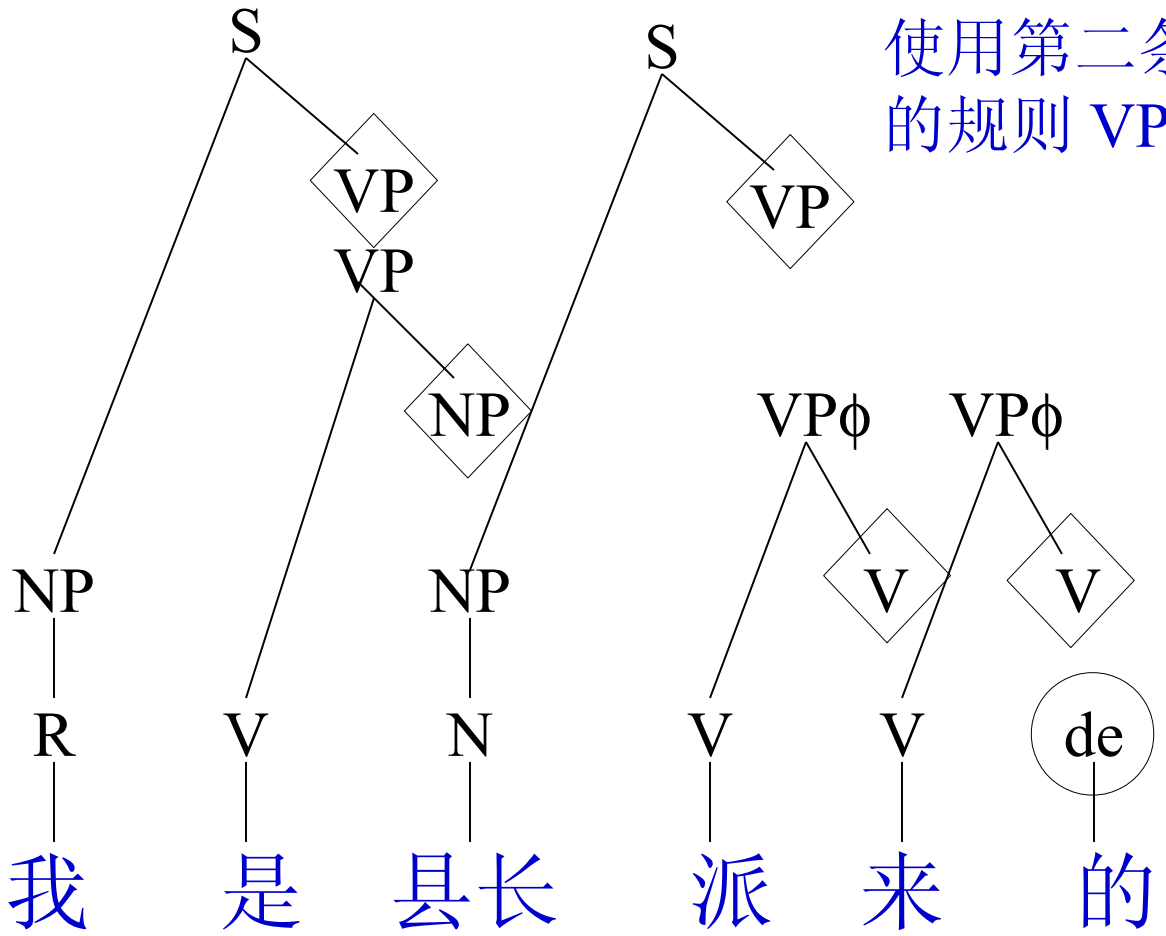
使用左角为 V 的规则
VP→V NP

左角分析法—示例 (21)



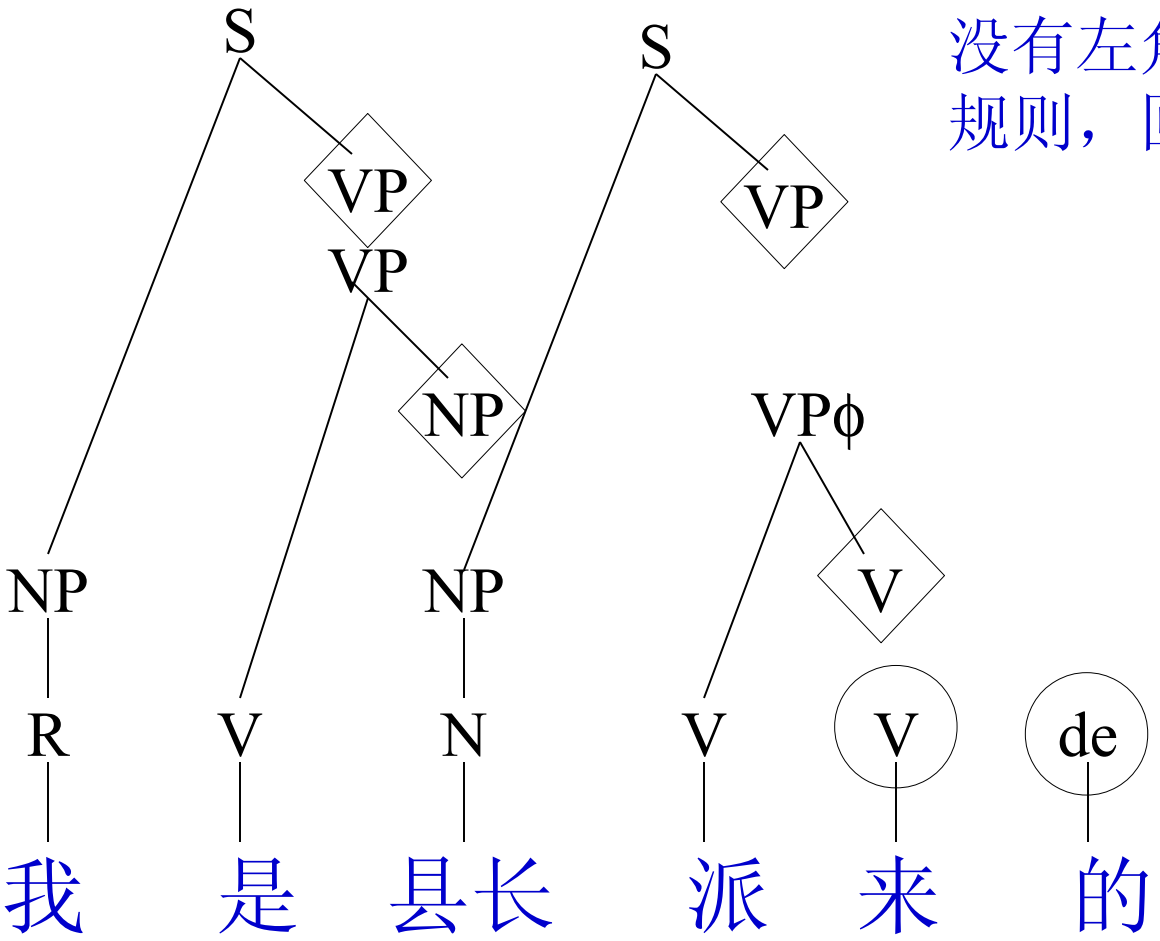
没有左角为 de 的规则，回溯

左角分析法—示例 (20)



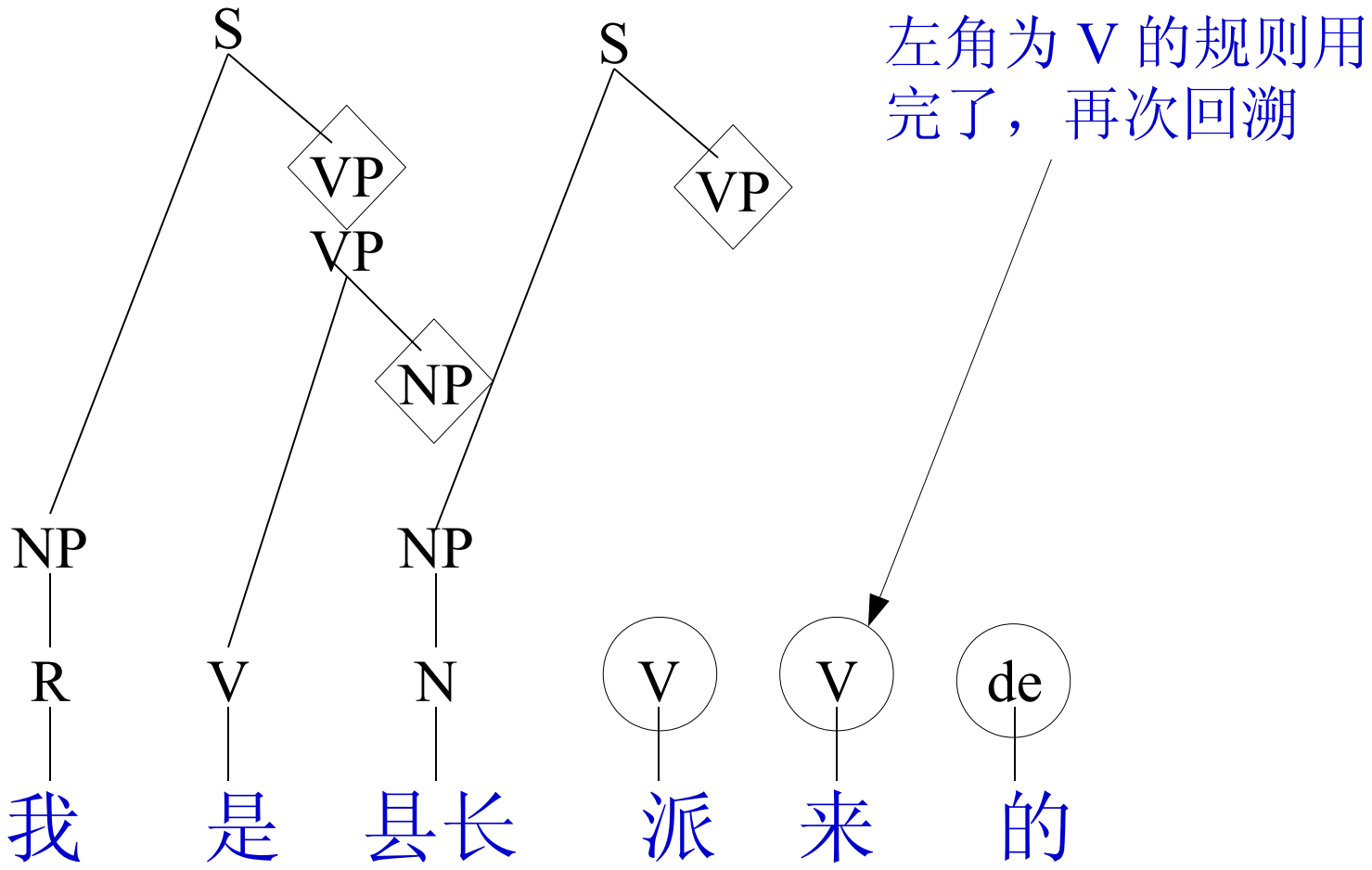
使用第二条左角为 V 的规则 VPϕ →V V

左角分析法—示例 (21)

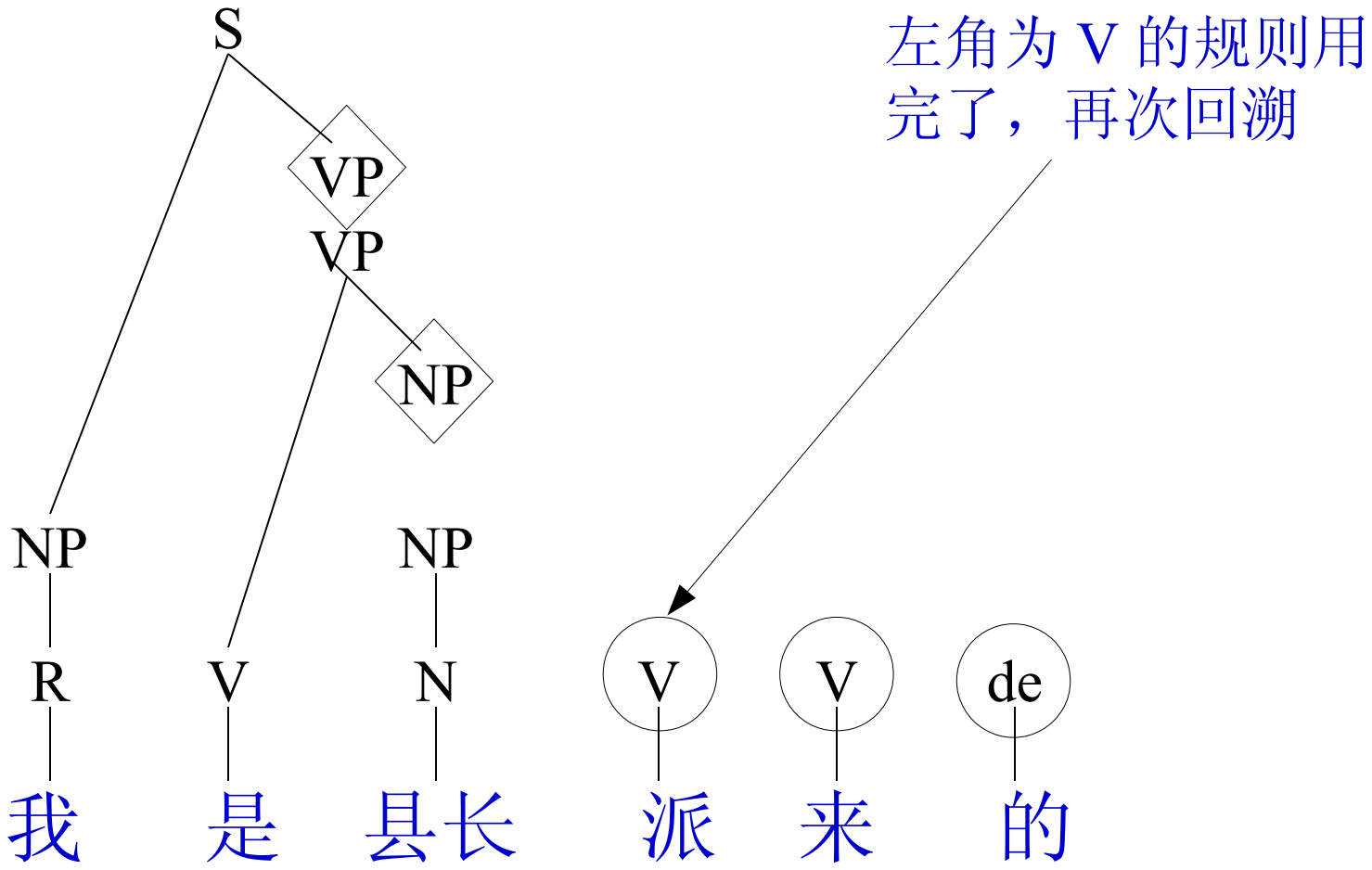


没有左角为 de 的规则，回溯

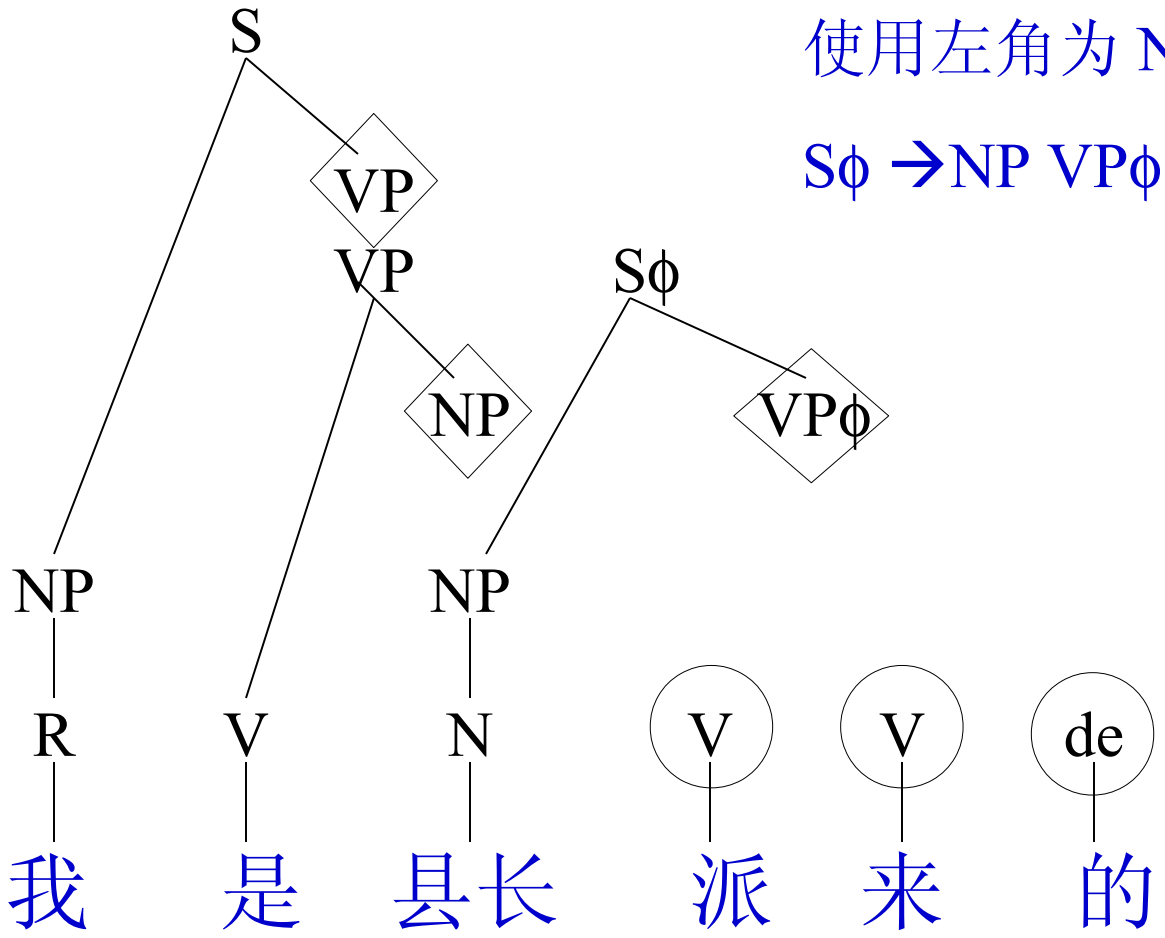
左角分析法—示例 (22)



左角分析法—示例 (23)



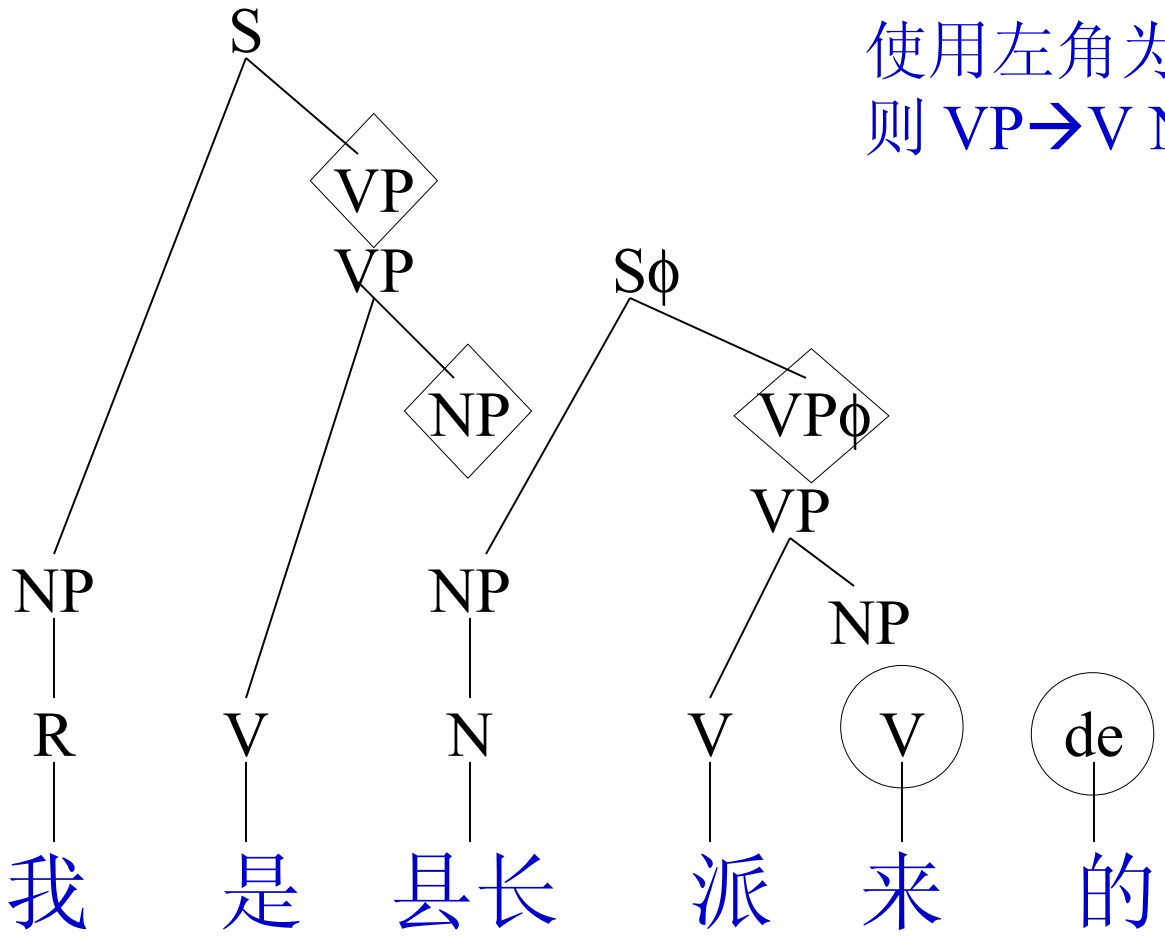
左角分析法一示例 (24)



使用左角为 NP 的规则

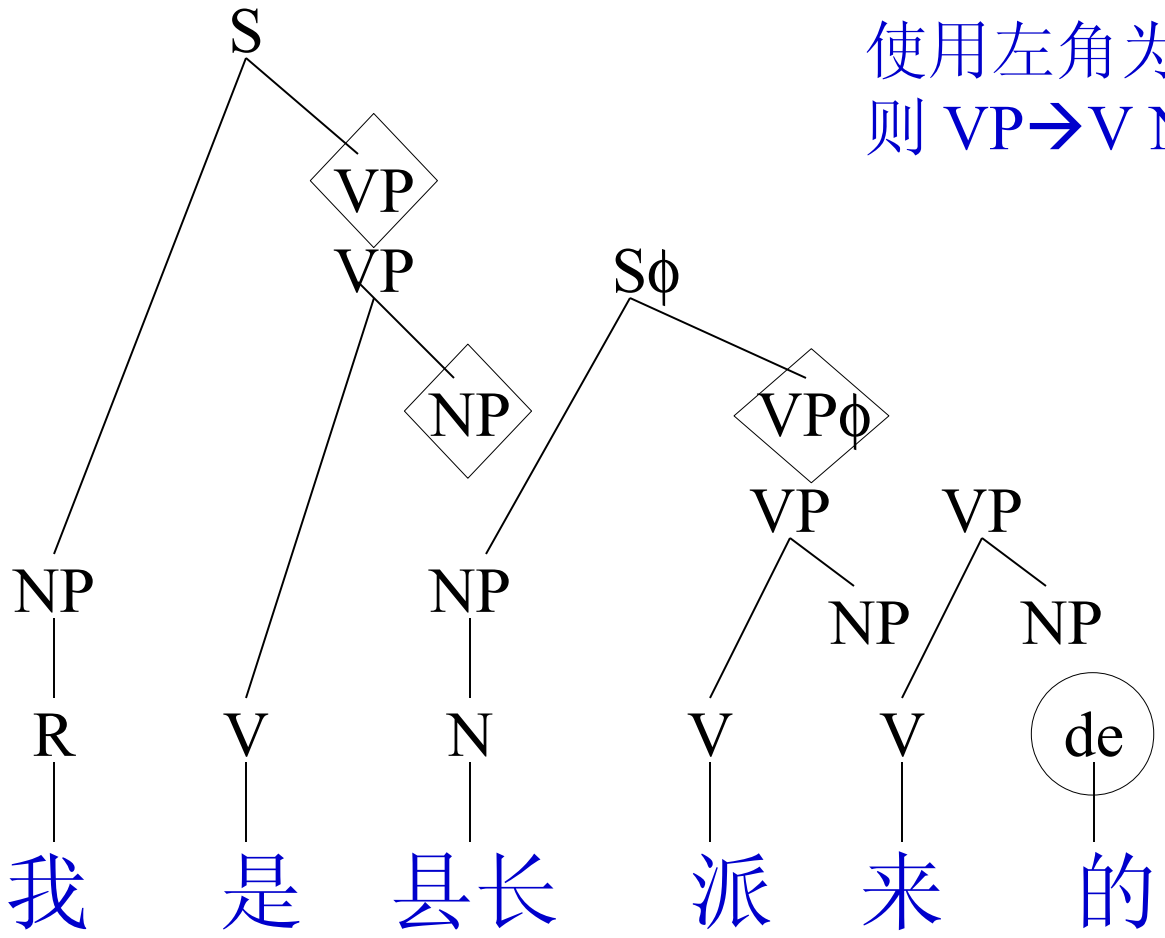
Sϕ →NP VPϕ

左角分析法—示例 (25)



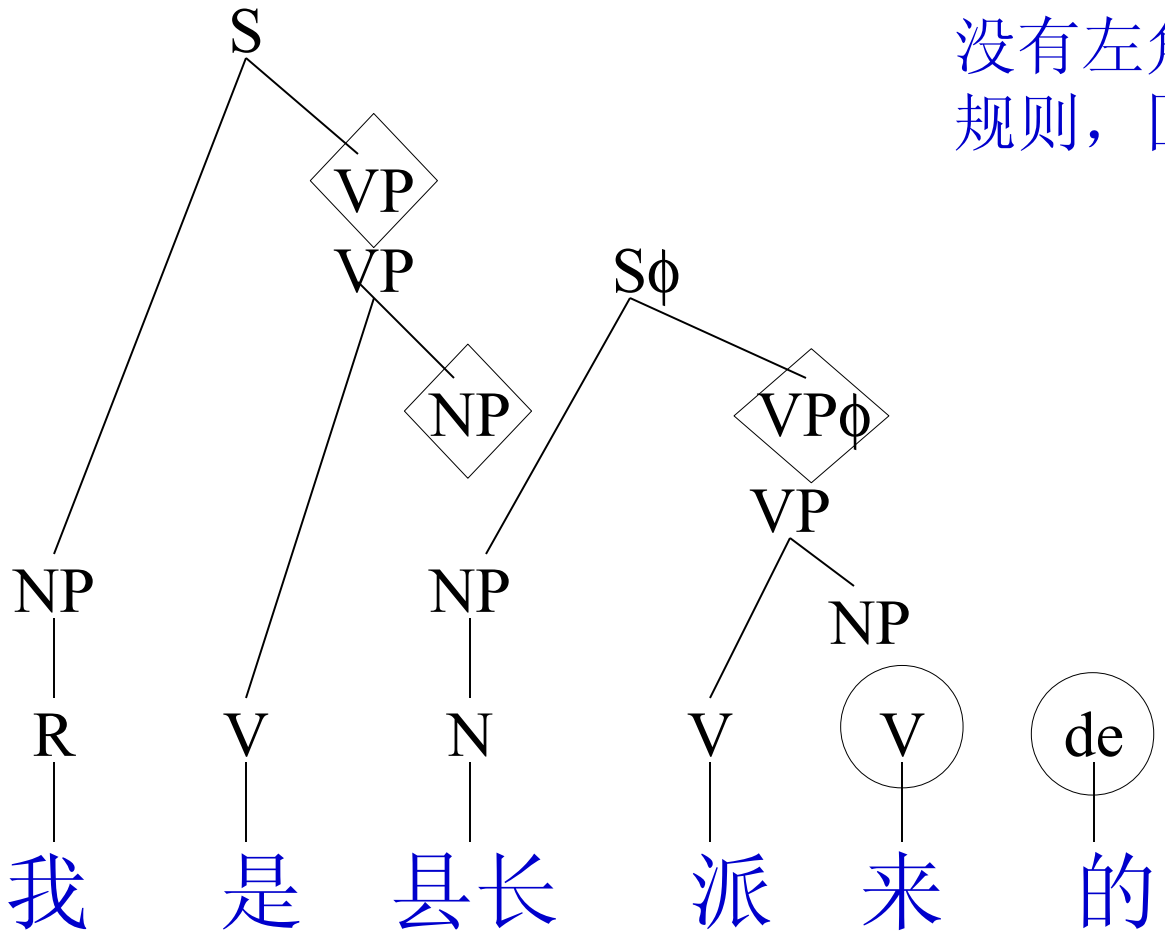
使用左角为 V 的规则 VP→V NP

左角分析法—示例 (26)



使用左角为 V 的规则 VP→V NP

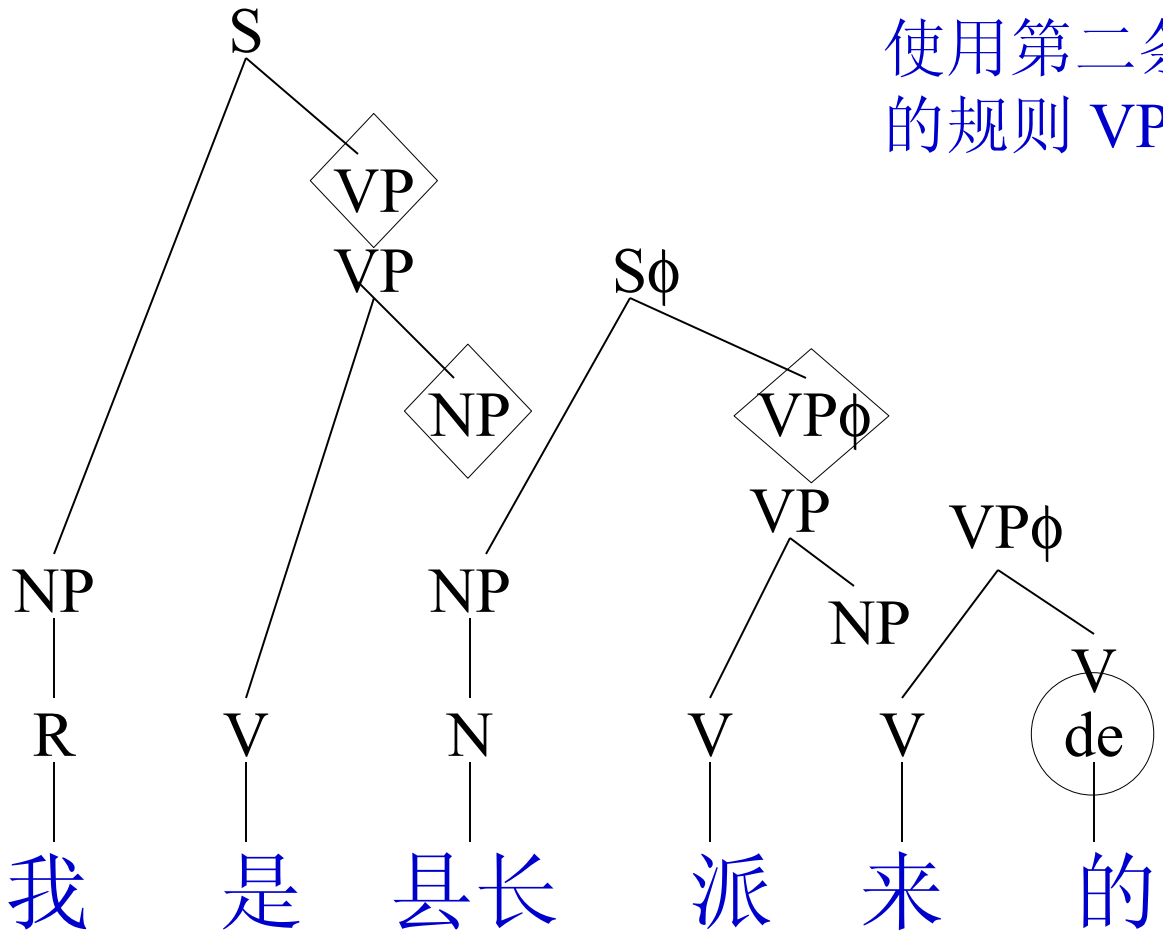
左角分析法—示例 (27)



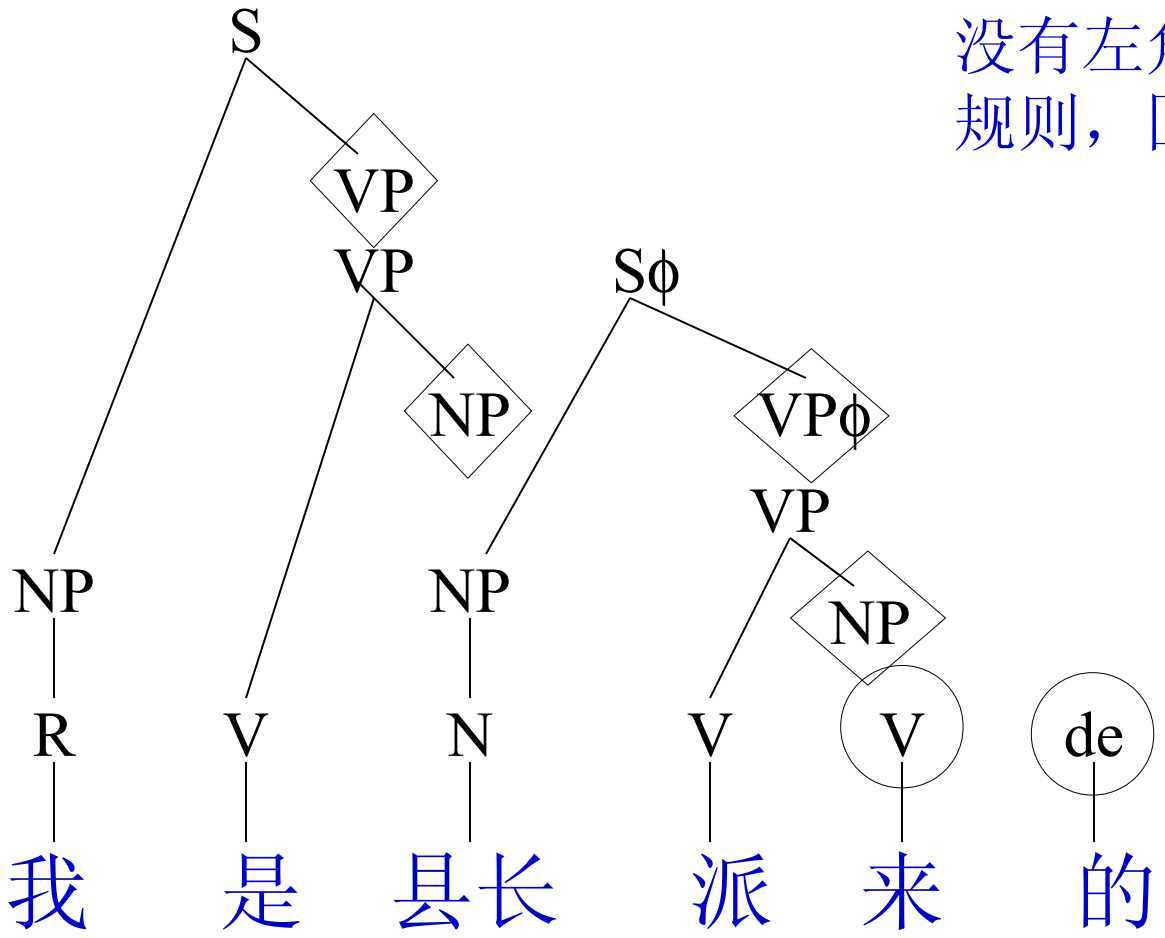
没有左角为 de 的规则，回溯

左角分析法—示例 (28)

使用第二条左角为 V 的规则 VPϕ →V V

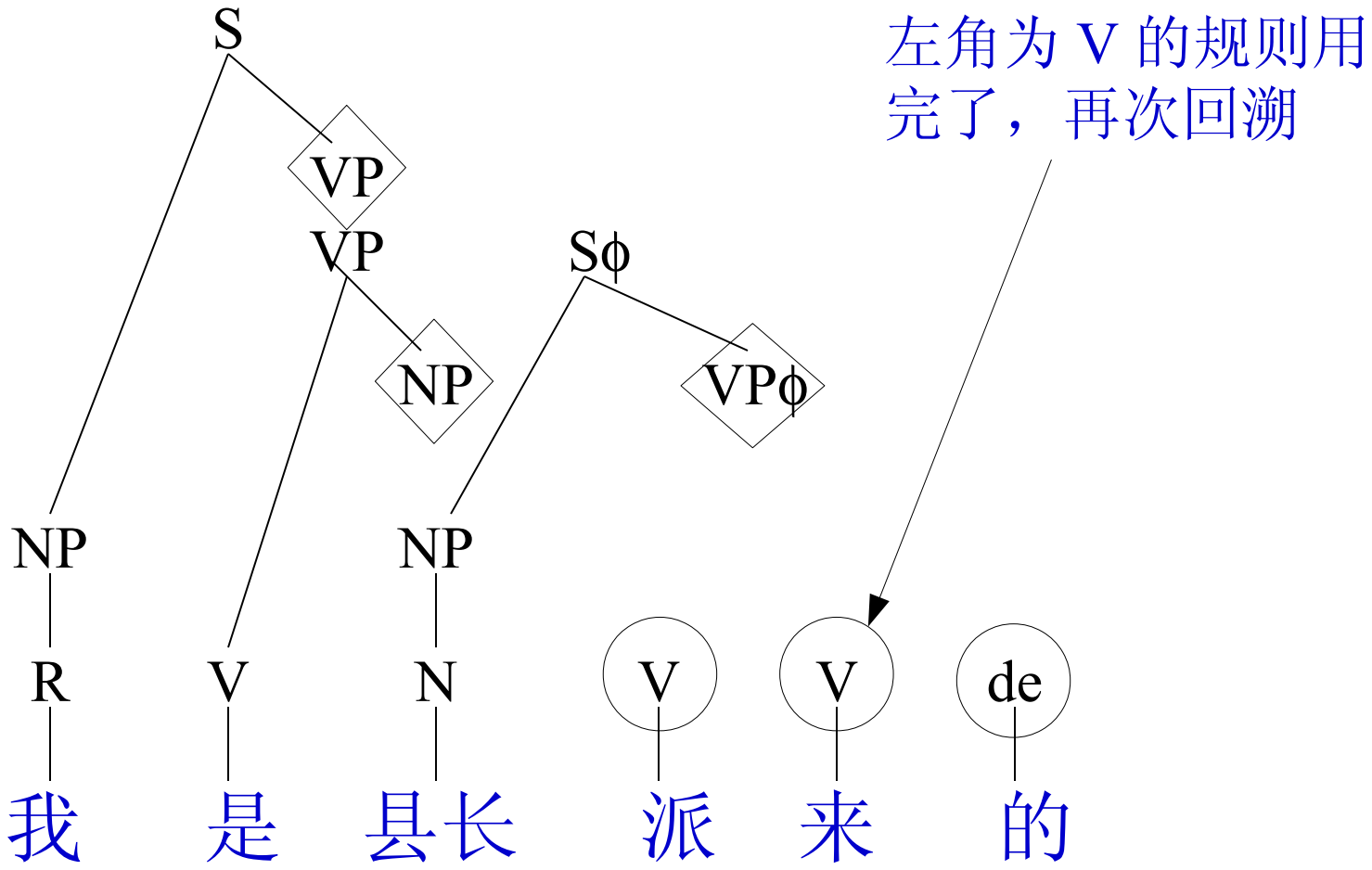


左角分析法—示例 (29)



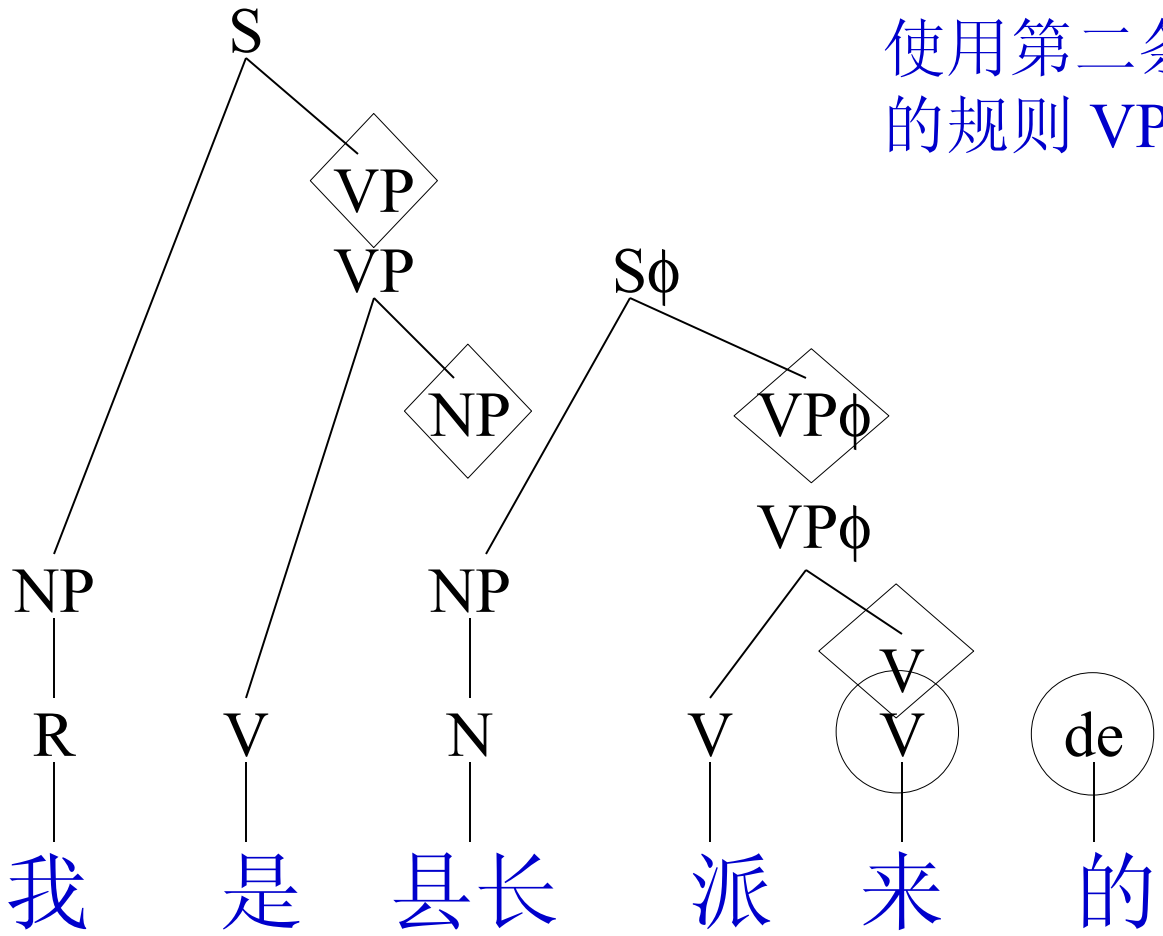
没有左角为 de 的规则，回溯

左角分析法—示例 (30)



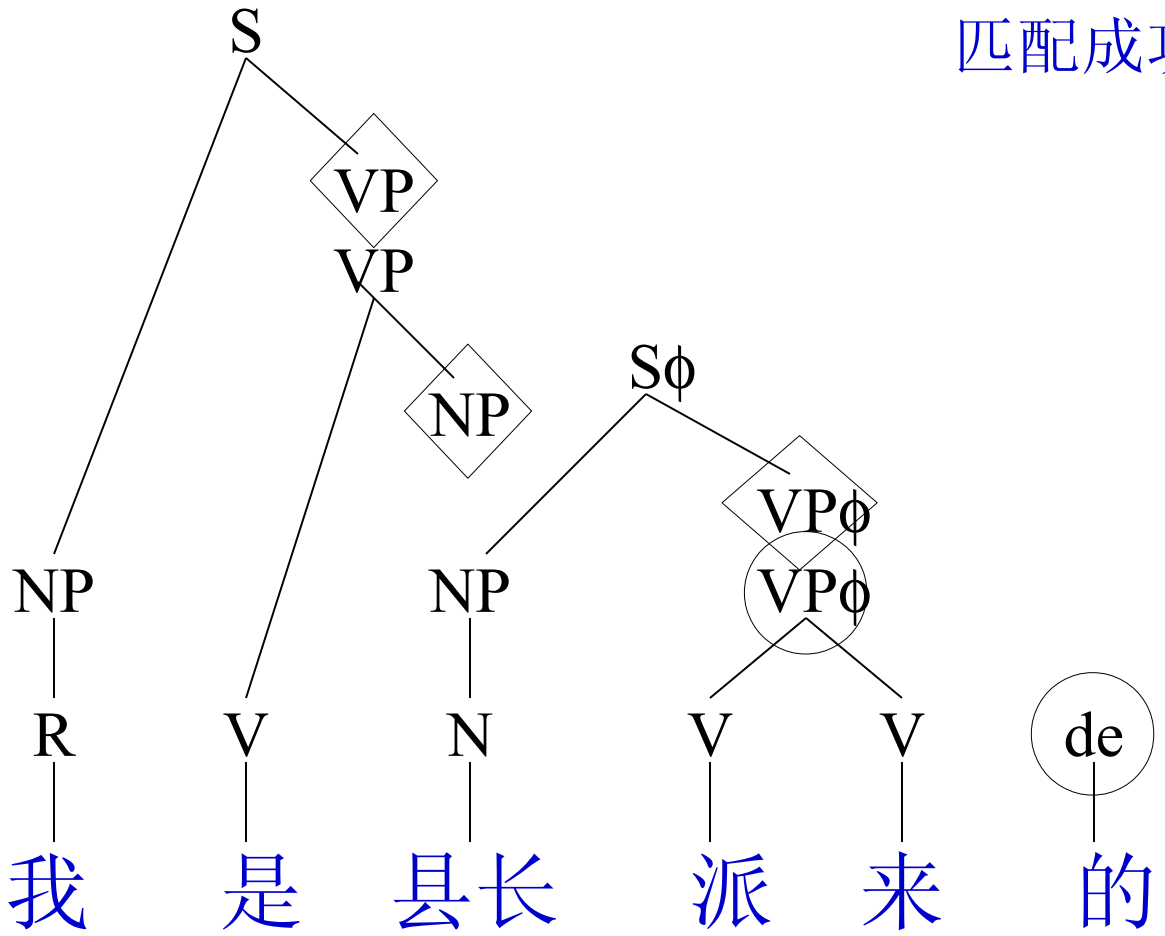
左角分析法—示例 (31)

使用第二条左角为 V 的规则 VPϕ →V V



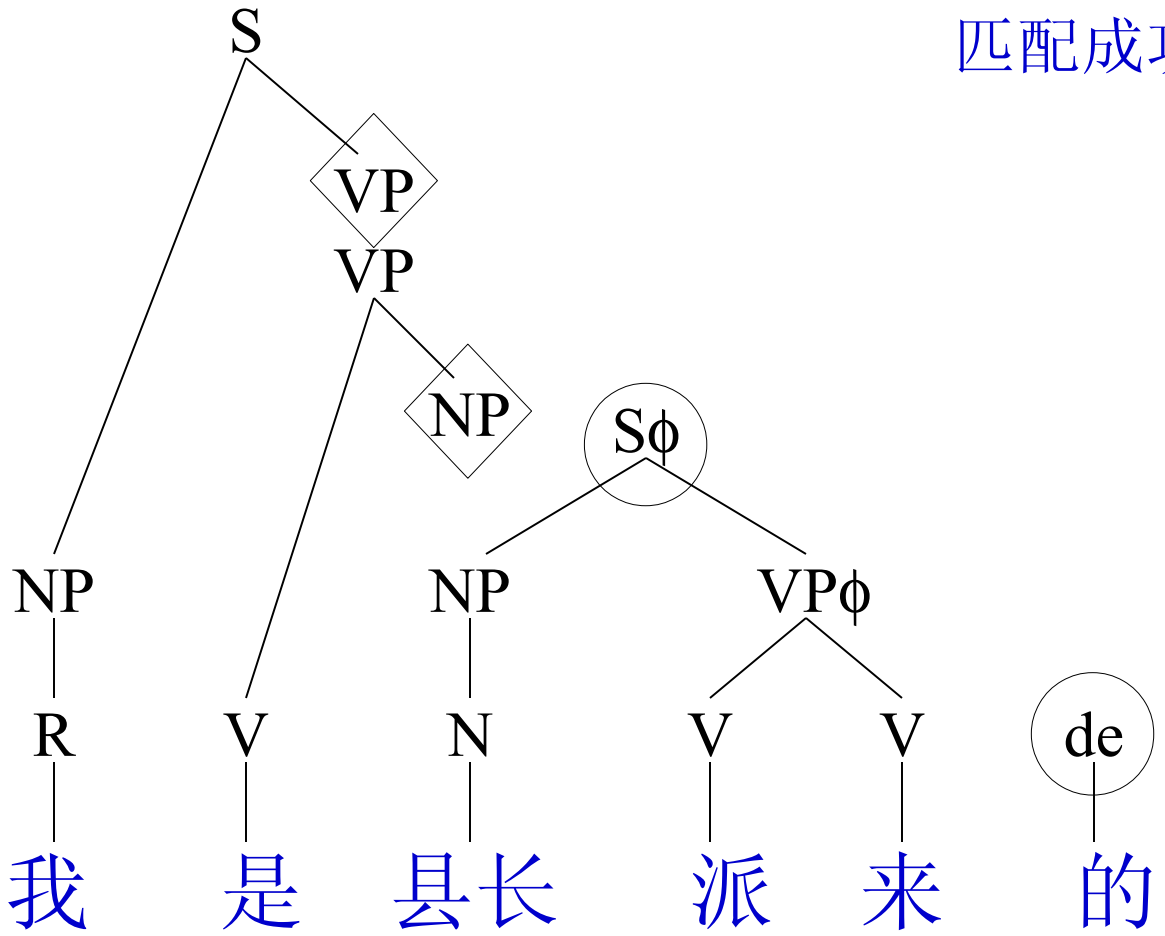
左角分析法—示例 (32)

匹配成功

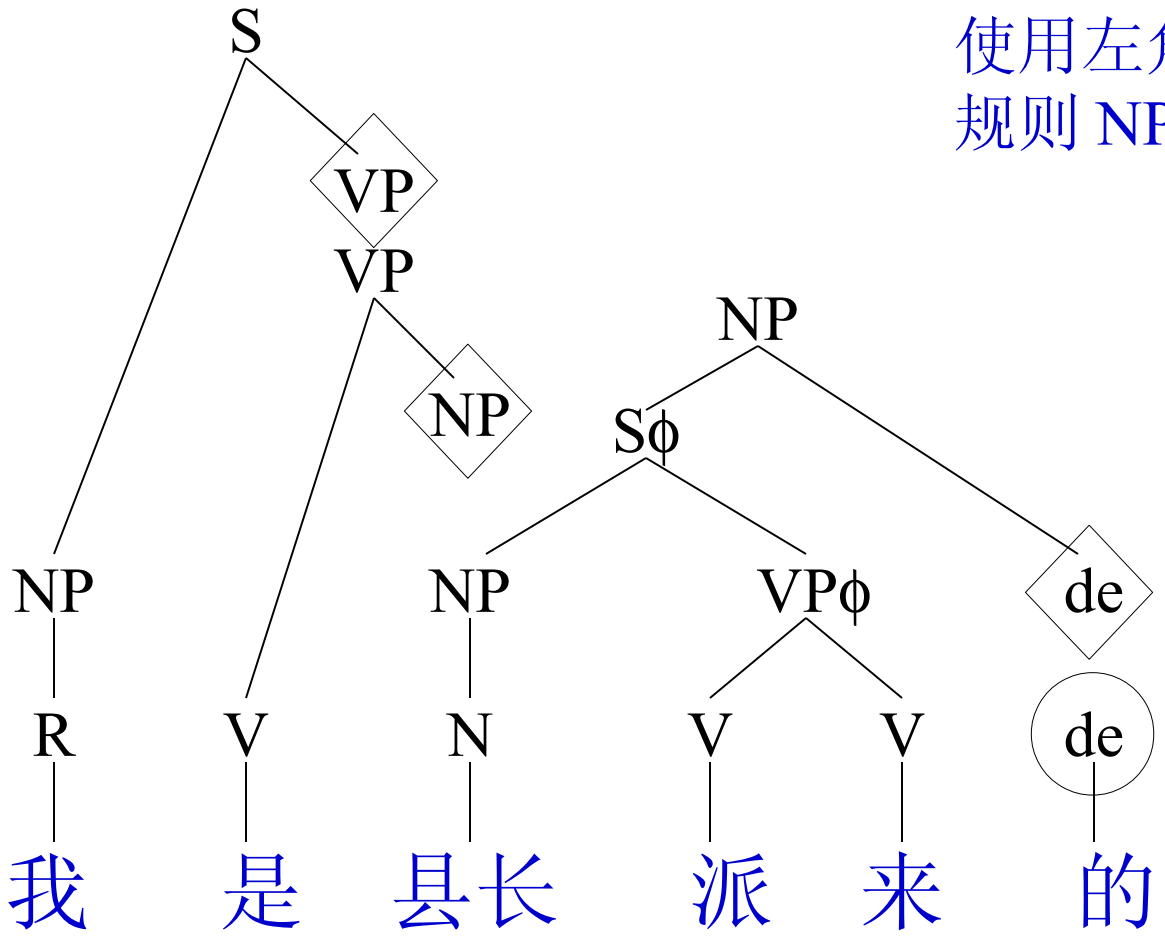


左角分析法—示例 (33)

匹配成功



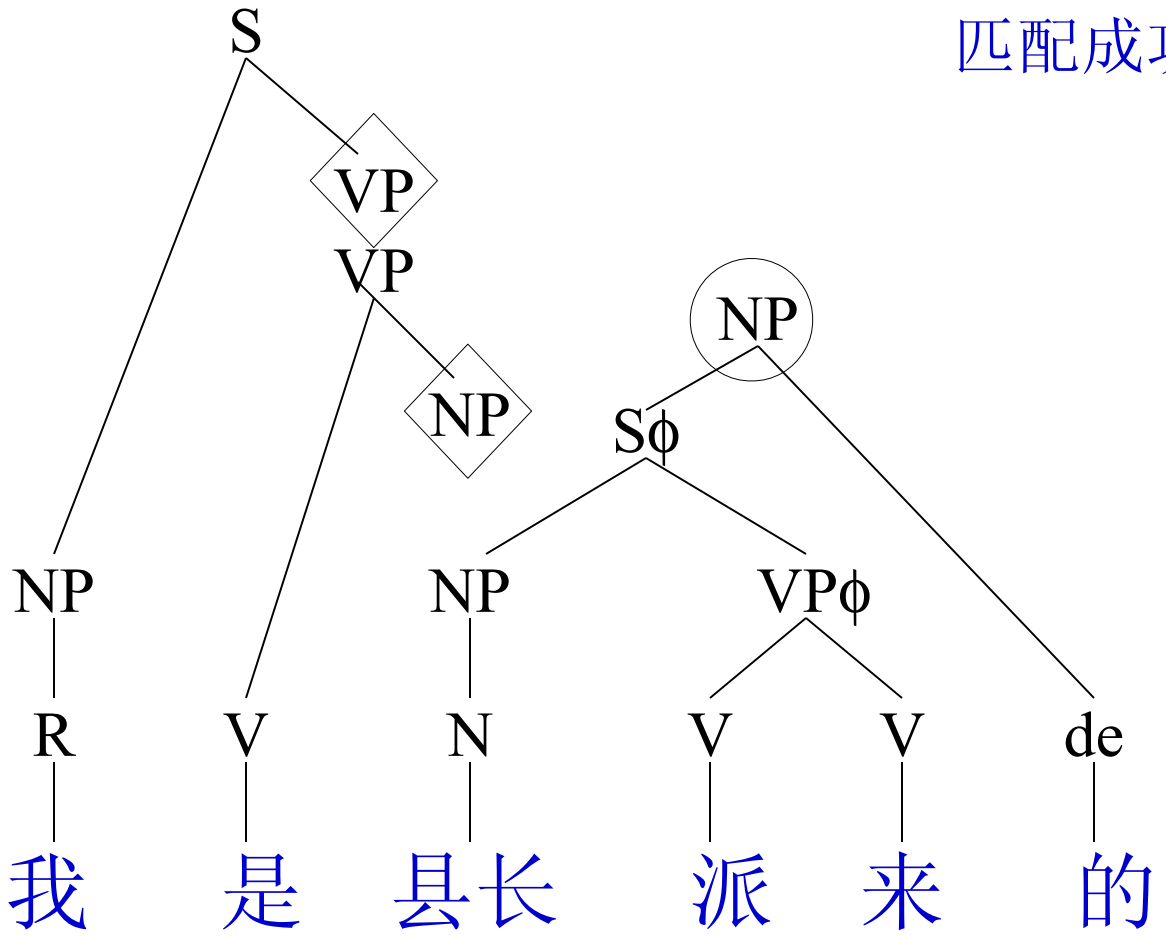
左角分析法—示例 (34)



使用左角为 Sϕ 的规则 NP → Sϕ de

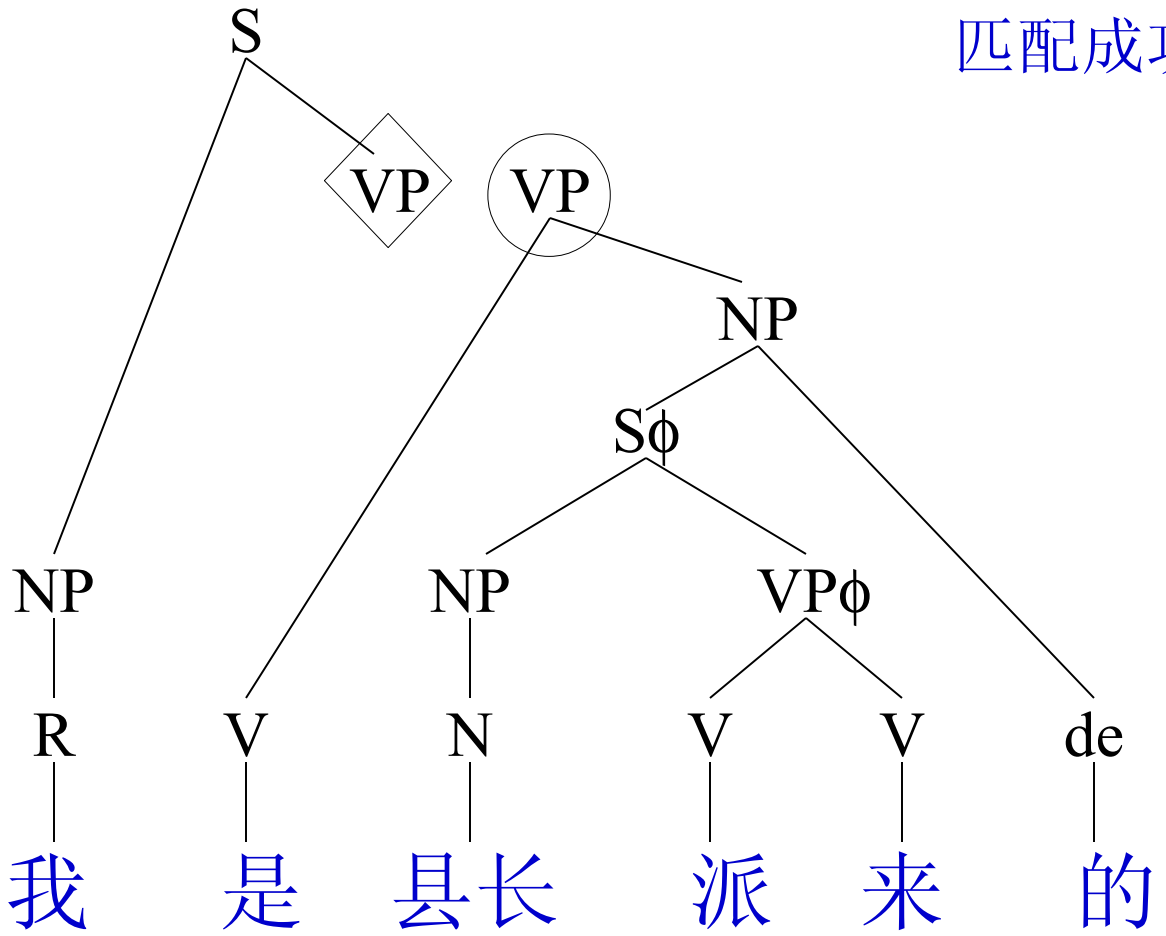
左角分析法—示例 (35)

匹配成功

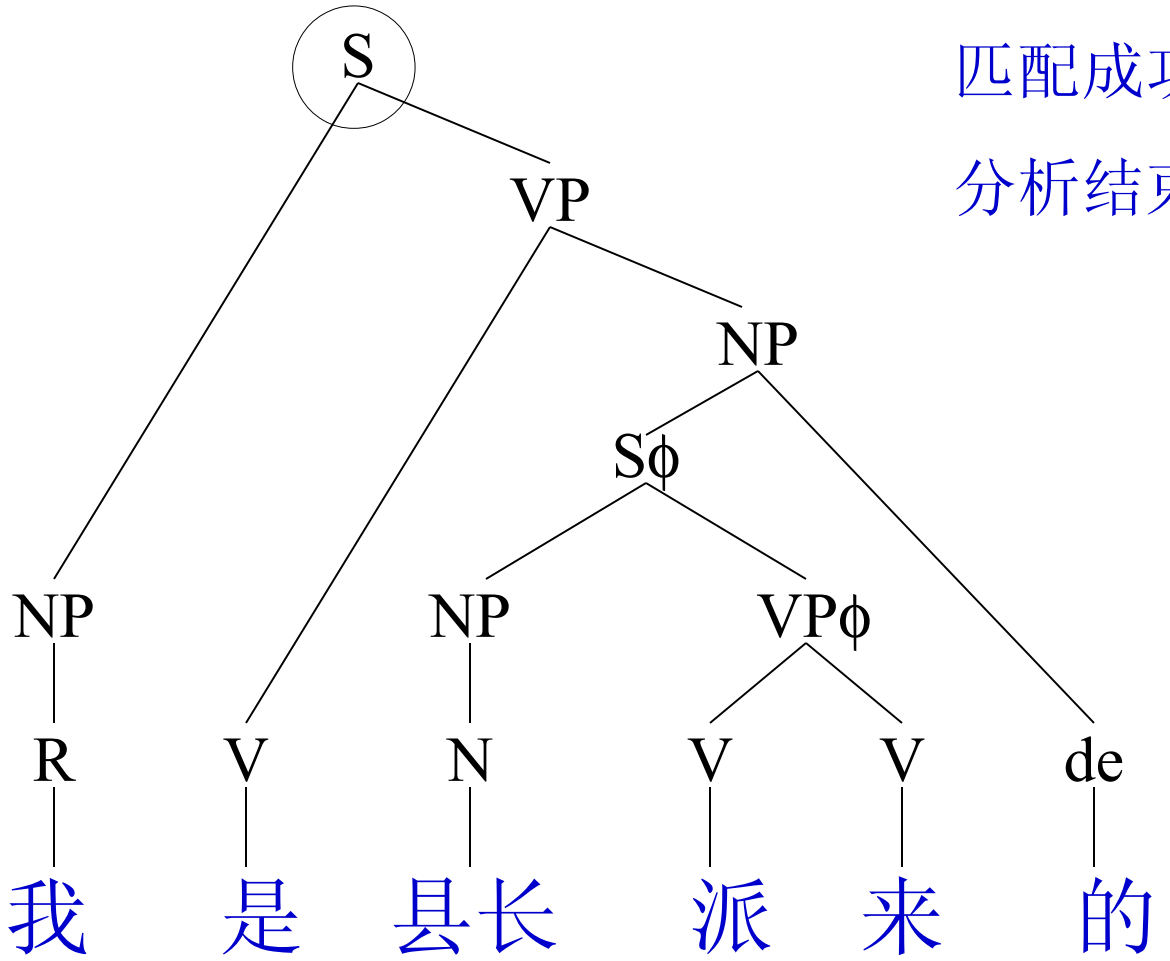


左角分析法—示例 (36)

匹配成功



左角分析法—示例 (37)



匹配成功

分析结束

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

内容提要


句法分析算法
什么是句法分析
句法分析策略
自顶向下
自底向上
左角
句法分析算法
移进-归约算法
CYK算法
Earley算法
Chart算法
Tomita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：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

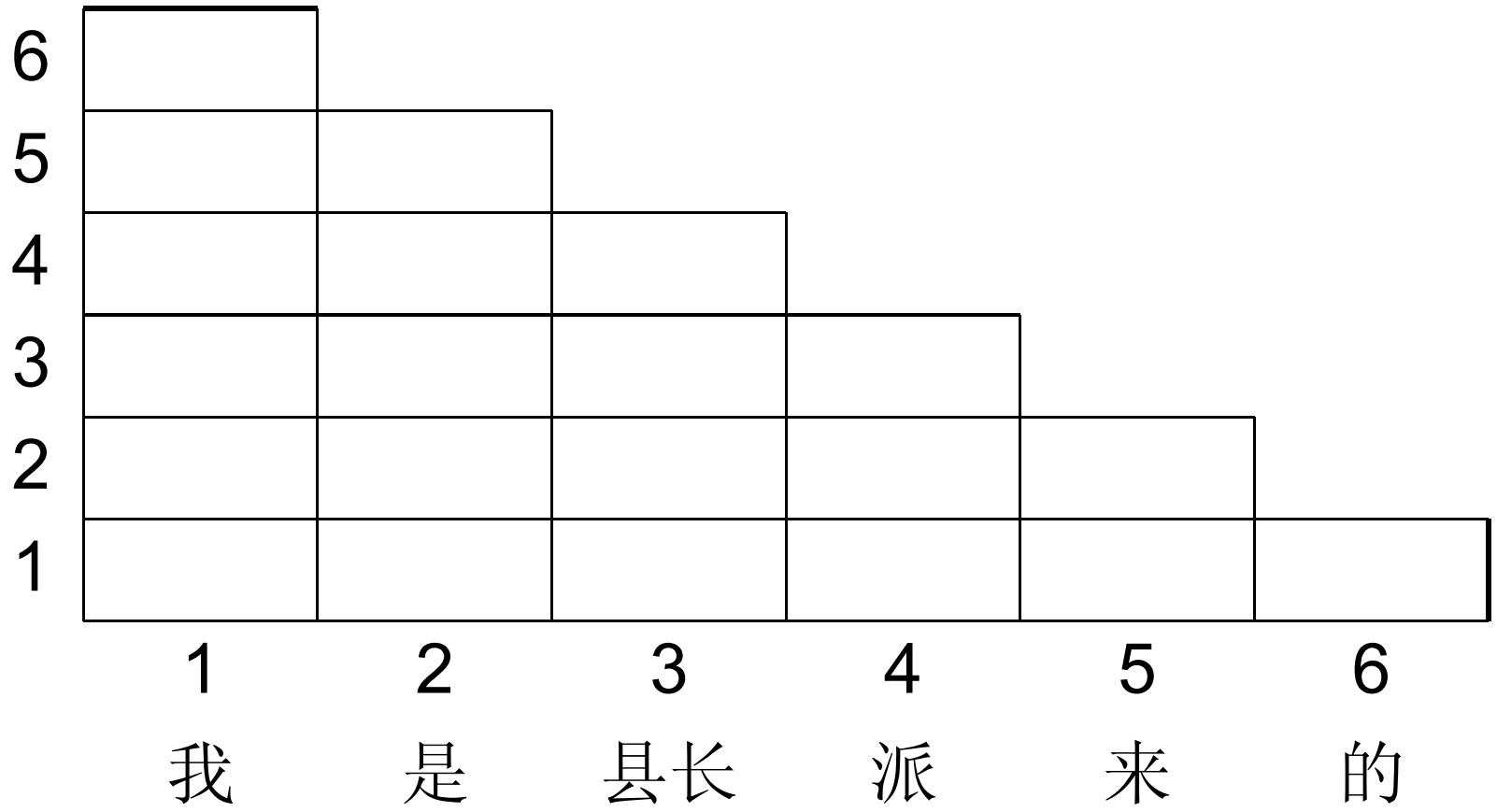
移进－归约算法的改进

- 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

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

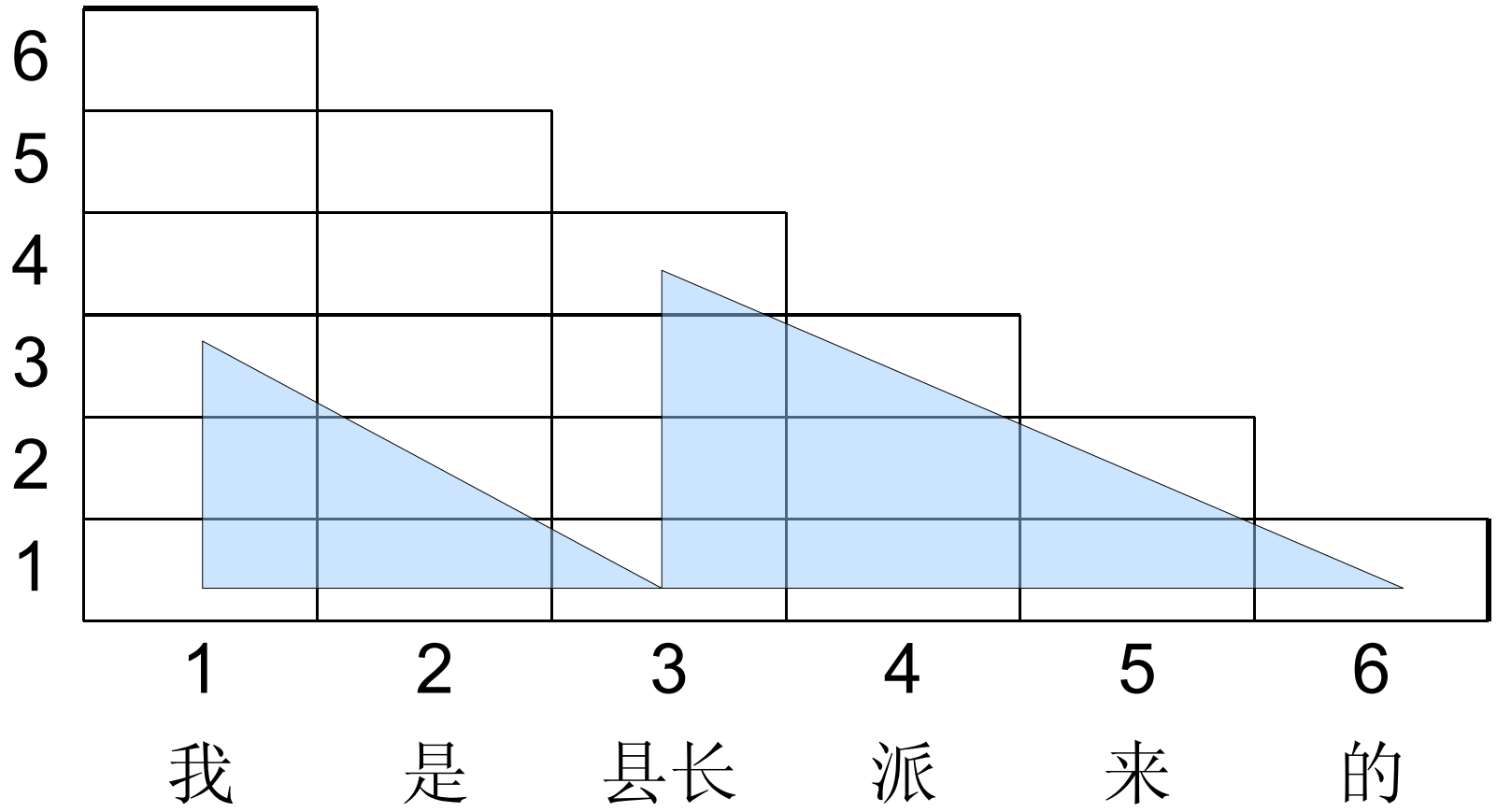
CYK 算法—数据结构 (1)



CYK 算法—数据结构 (2)

- 一个斜角二维矩阵：{ P(i , j) }
  - 每一个元素 P(i , j) 对应于输入句子中某一个区间（ 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

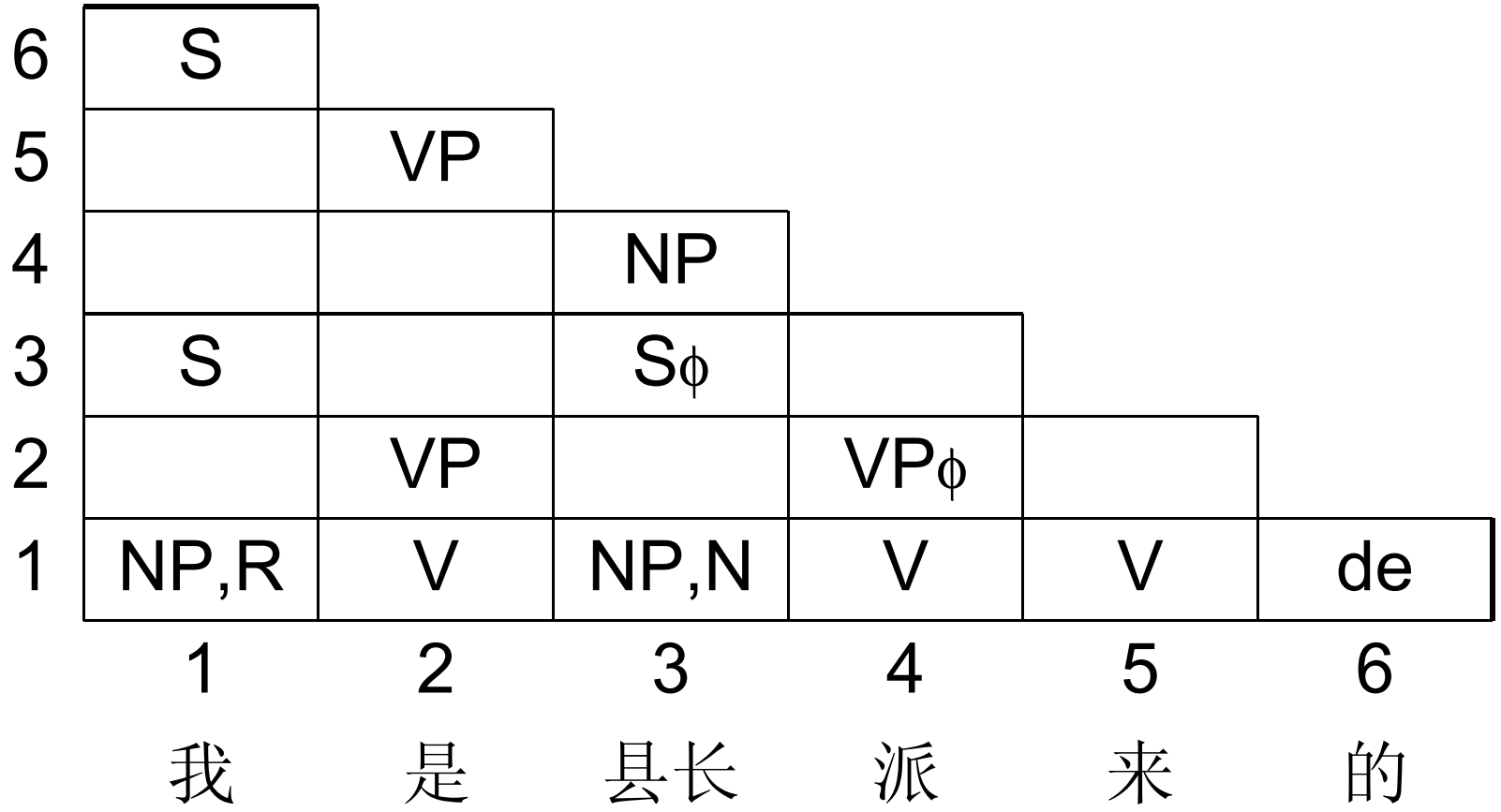
CYK 算法—数据结构 (3)



CYK 算法—数据结构 (4)

- 矩阵中填入该区间对应的词语序列上所有可能的短语标记

CYK 算法—数据结构 (5)



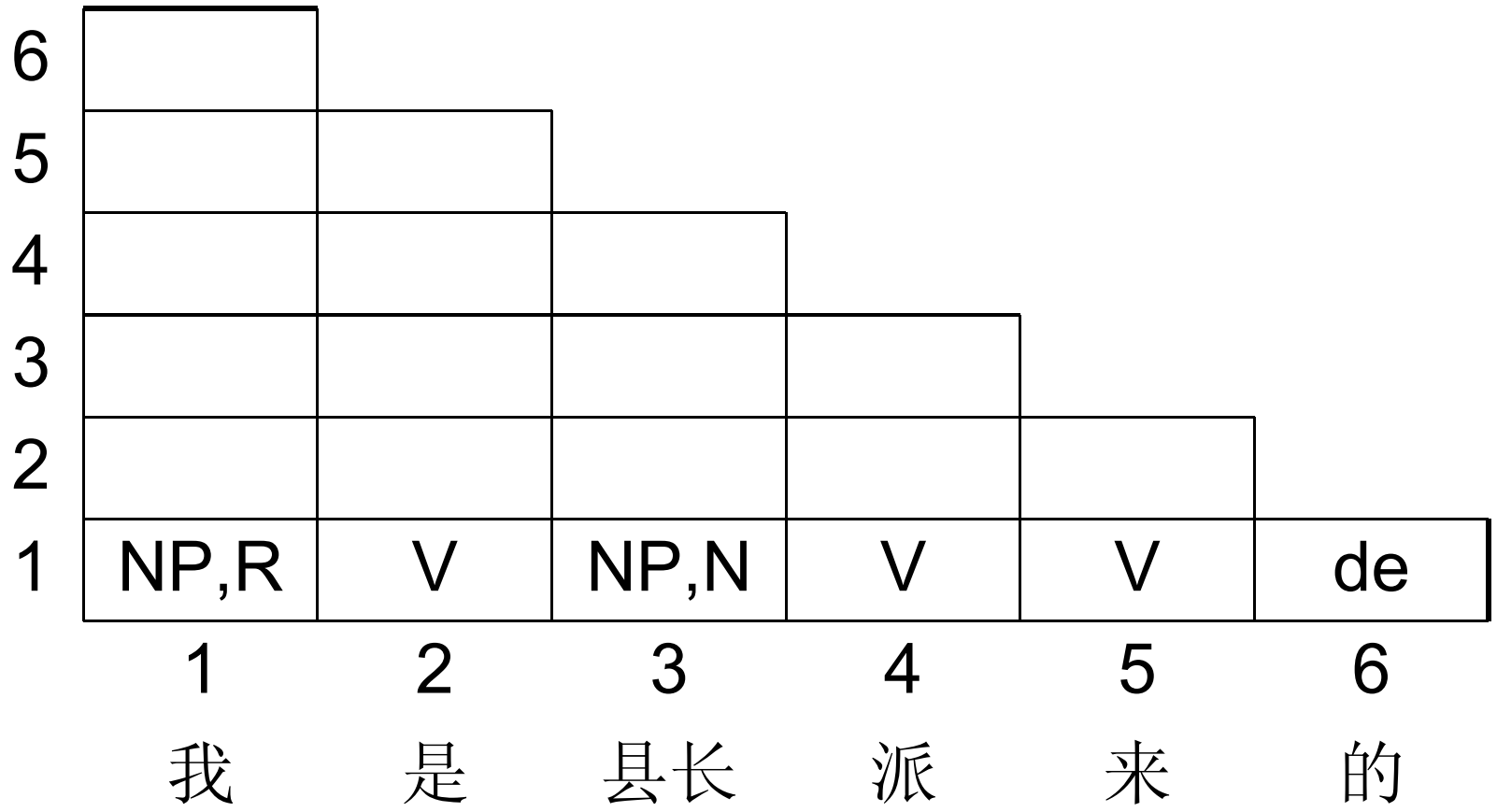
CYK 算法—数据结构 (6)

- 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

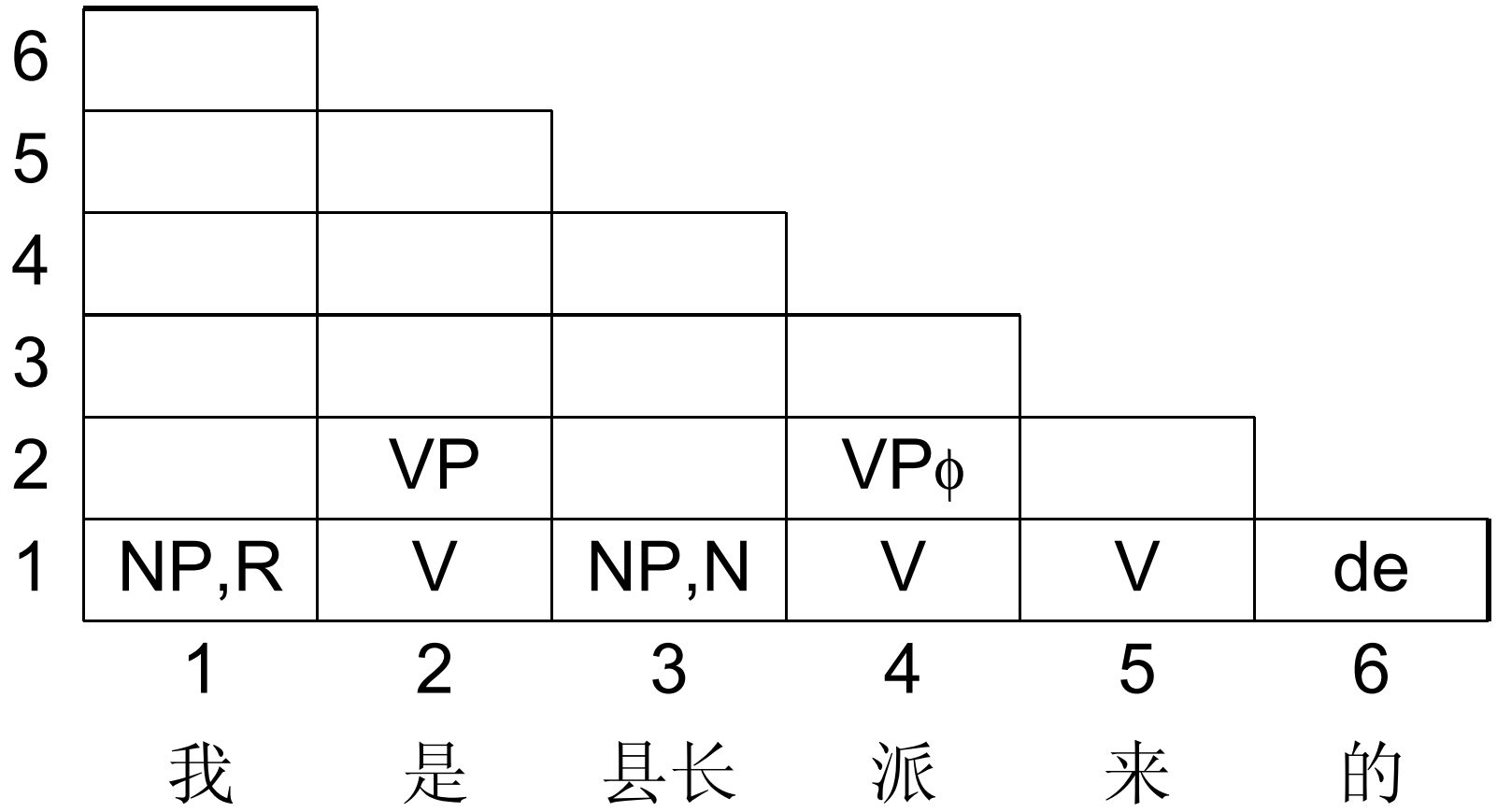
CYK 算法：算法描述

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

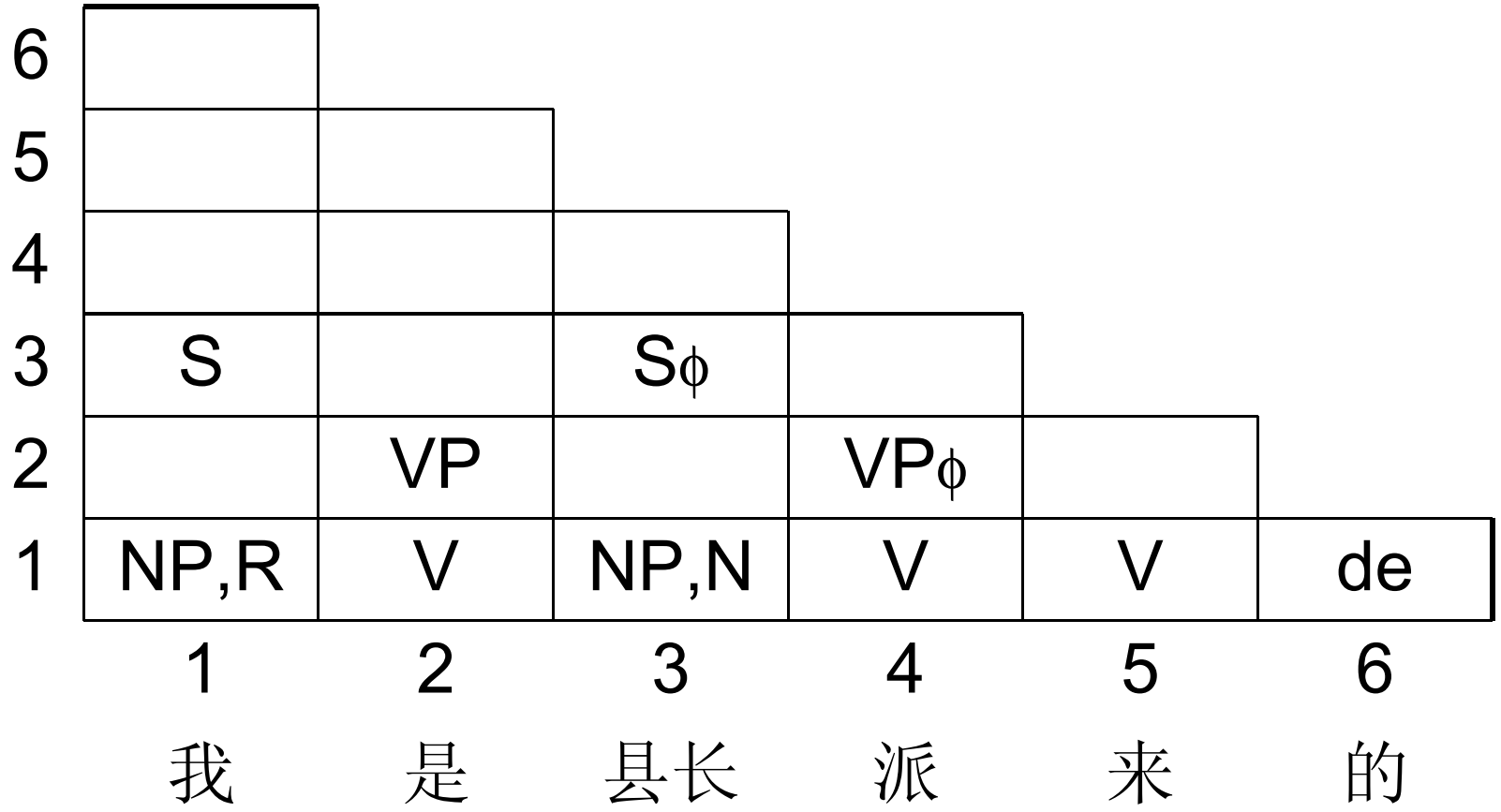
CYK 算法—运行示例 (1)



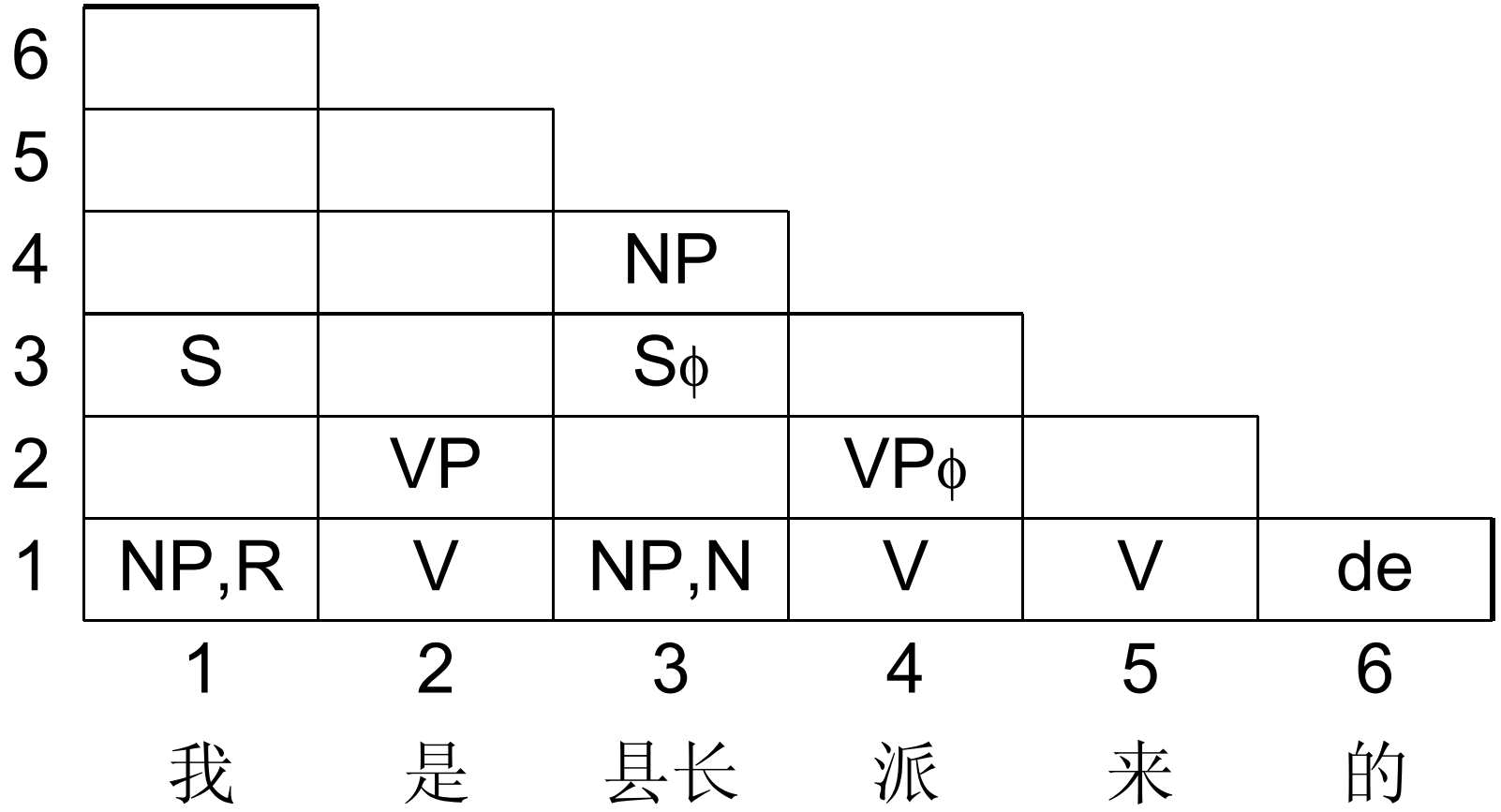
CYK 算法—运行示例 (2)



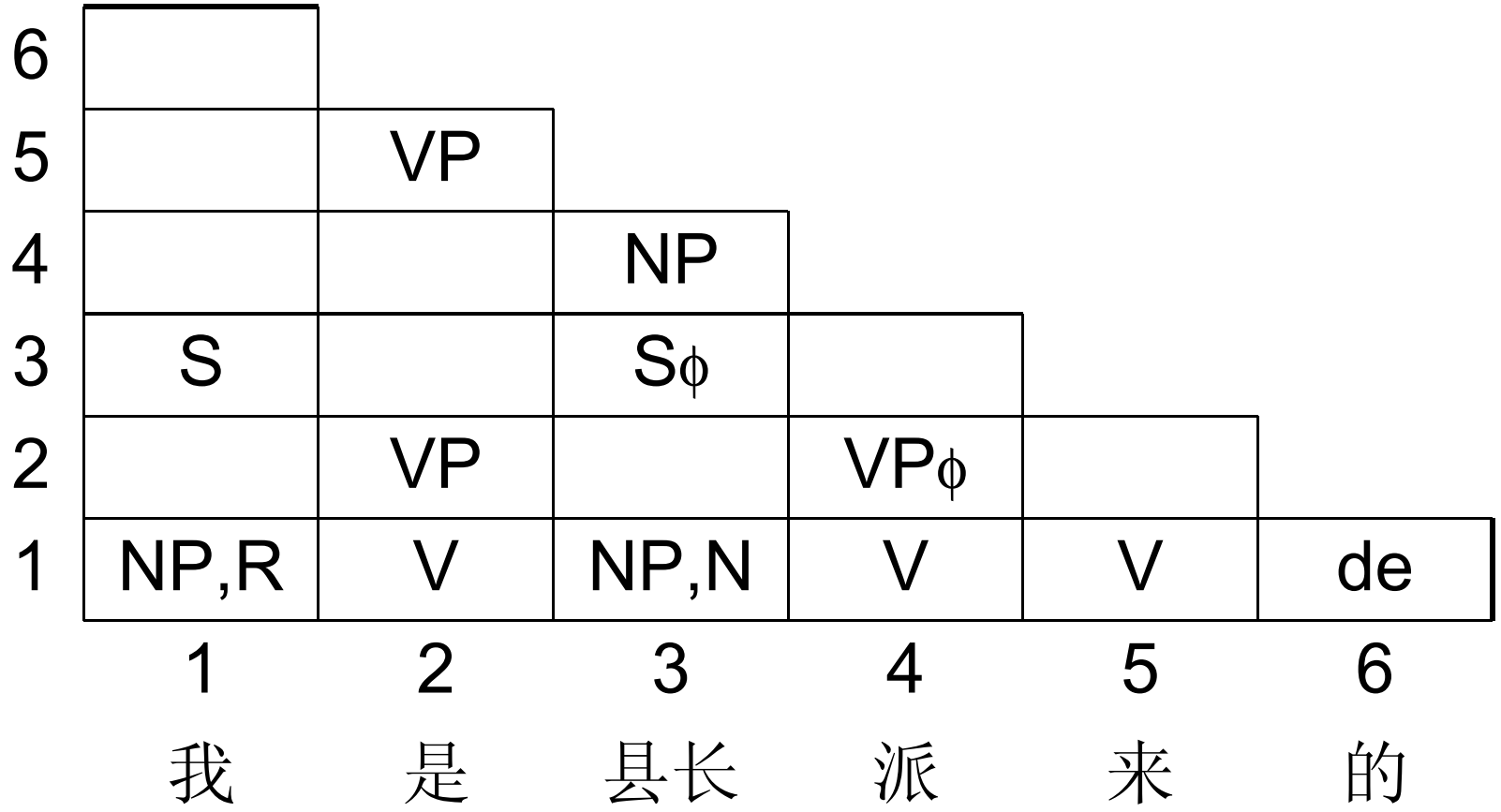
CYK 算法—运行示例 (3)



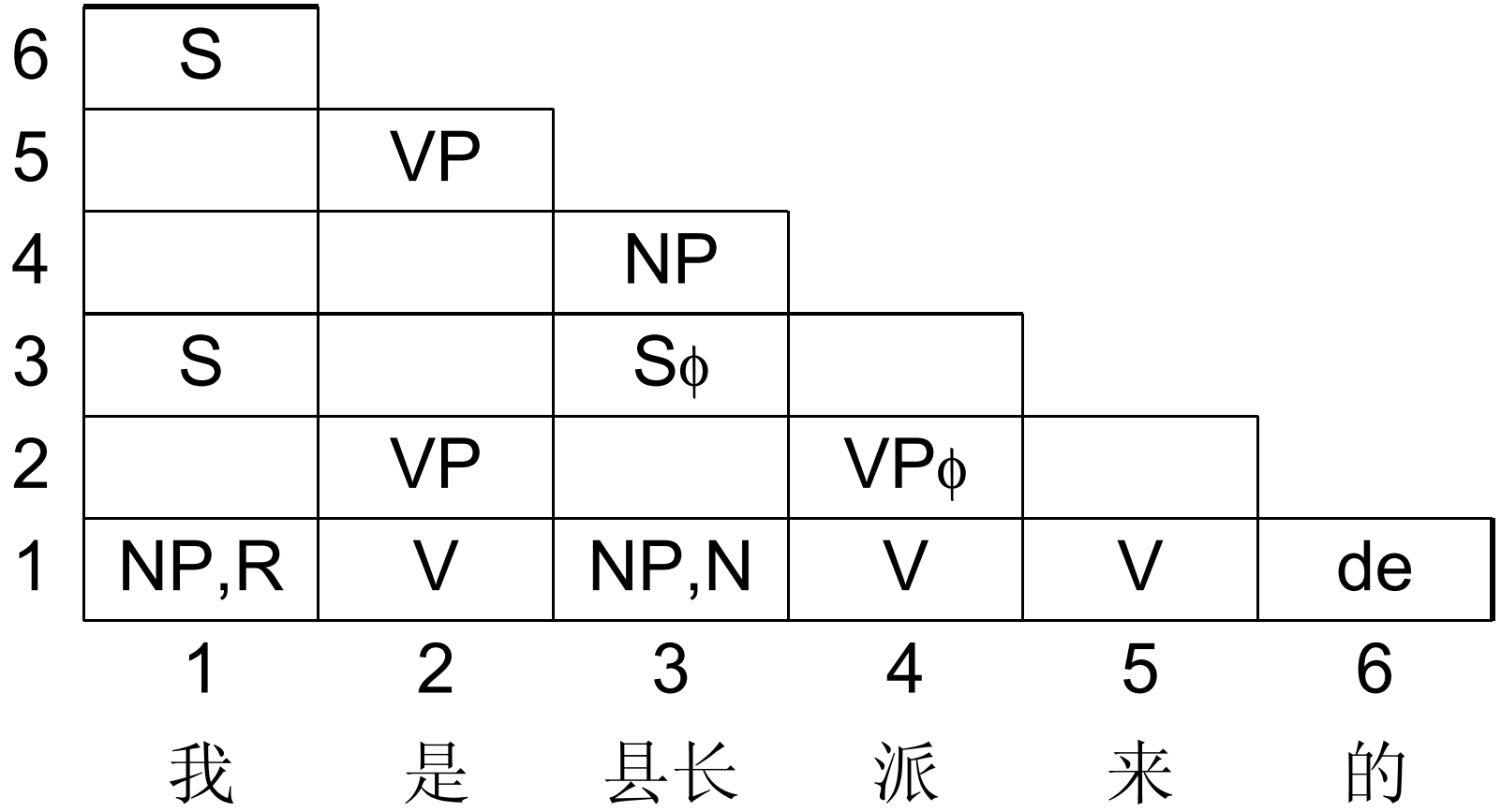
CYK 算法—运行示例 (4)



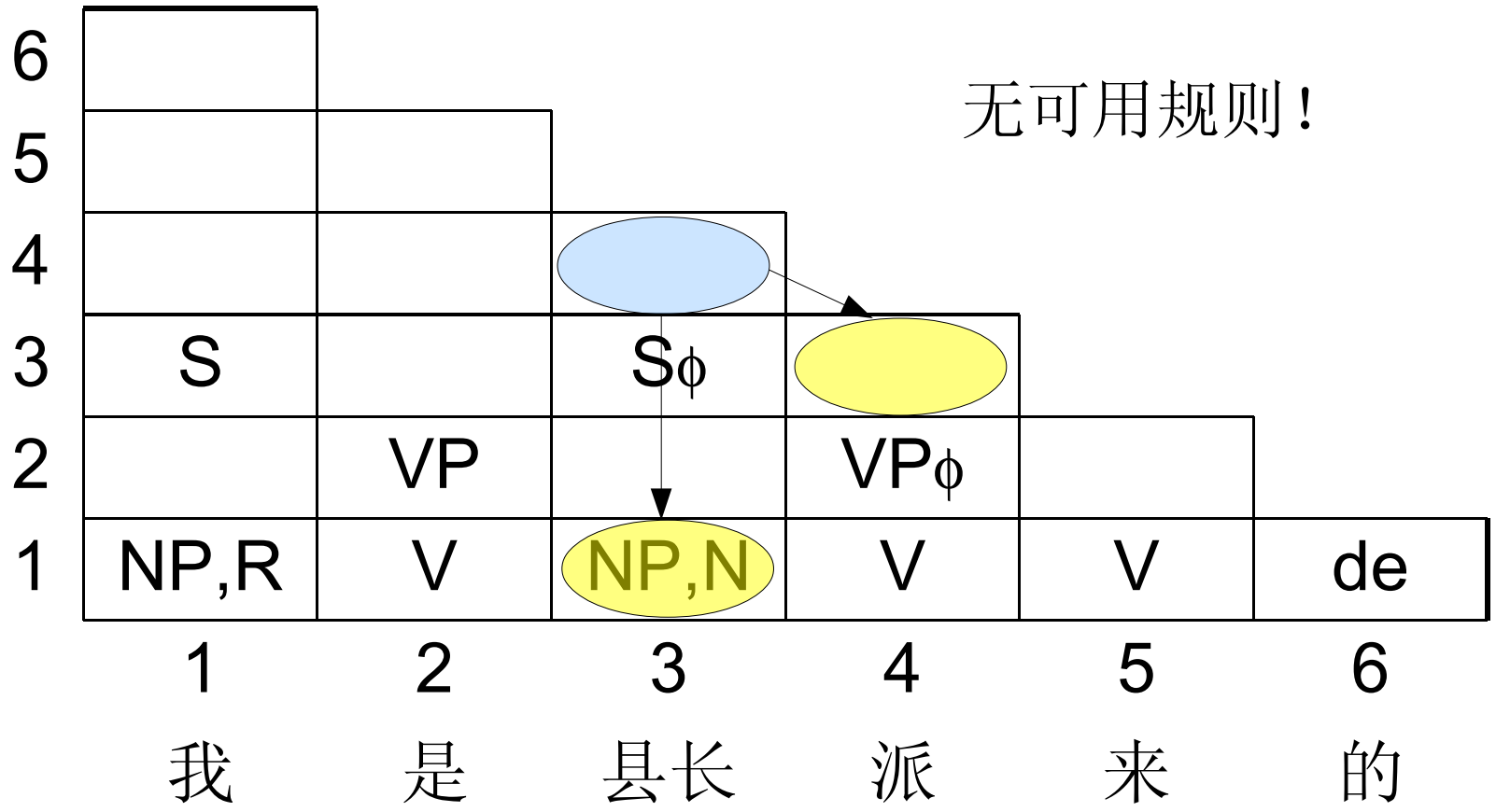
CYK 算法—运行示例 (5)



CYK 算法—运行示例 (6)

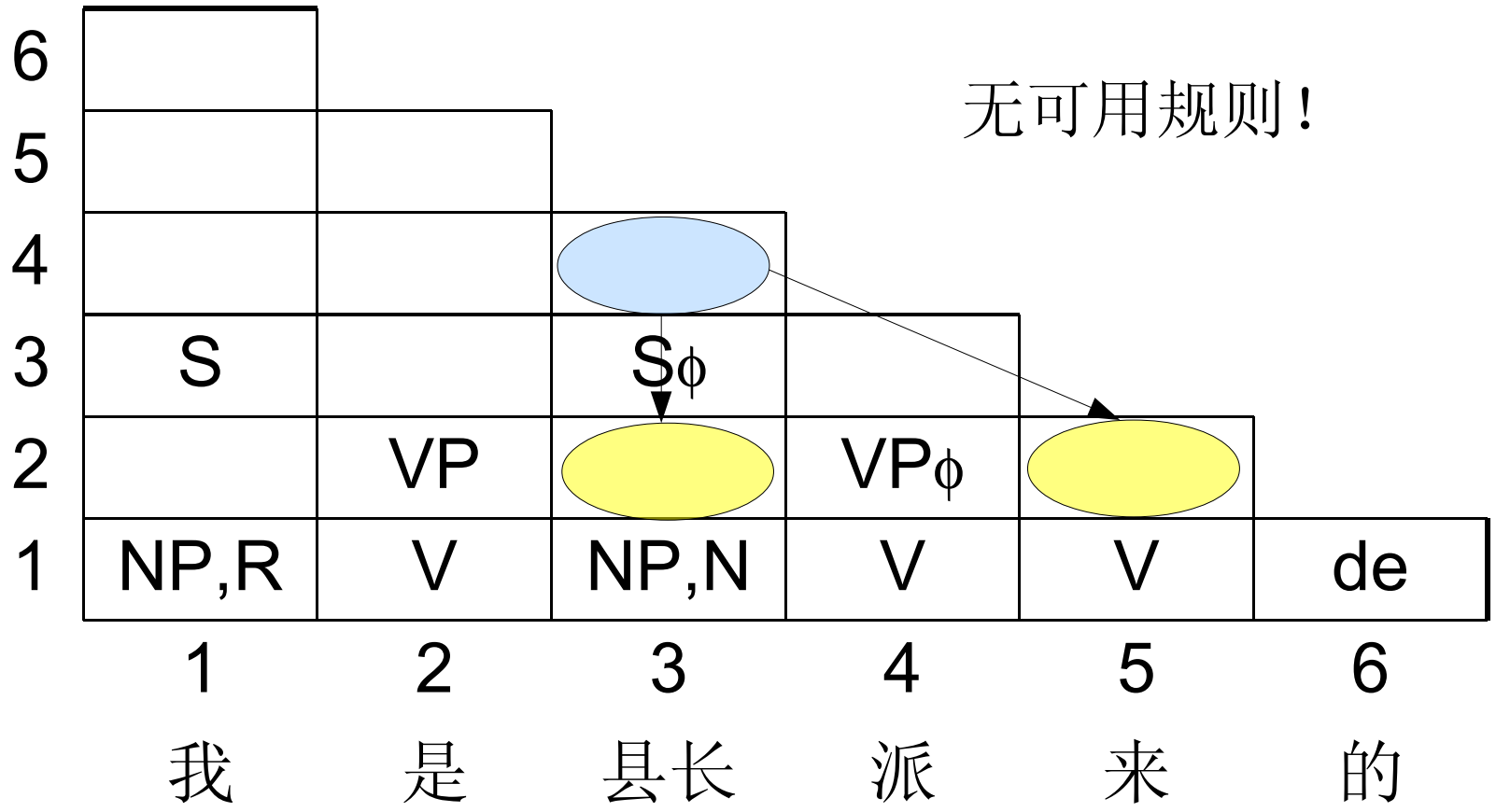


CYK 算法— Span 的划分 (1)

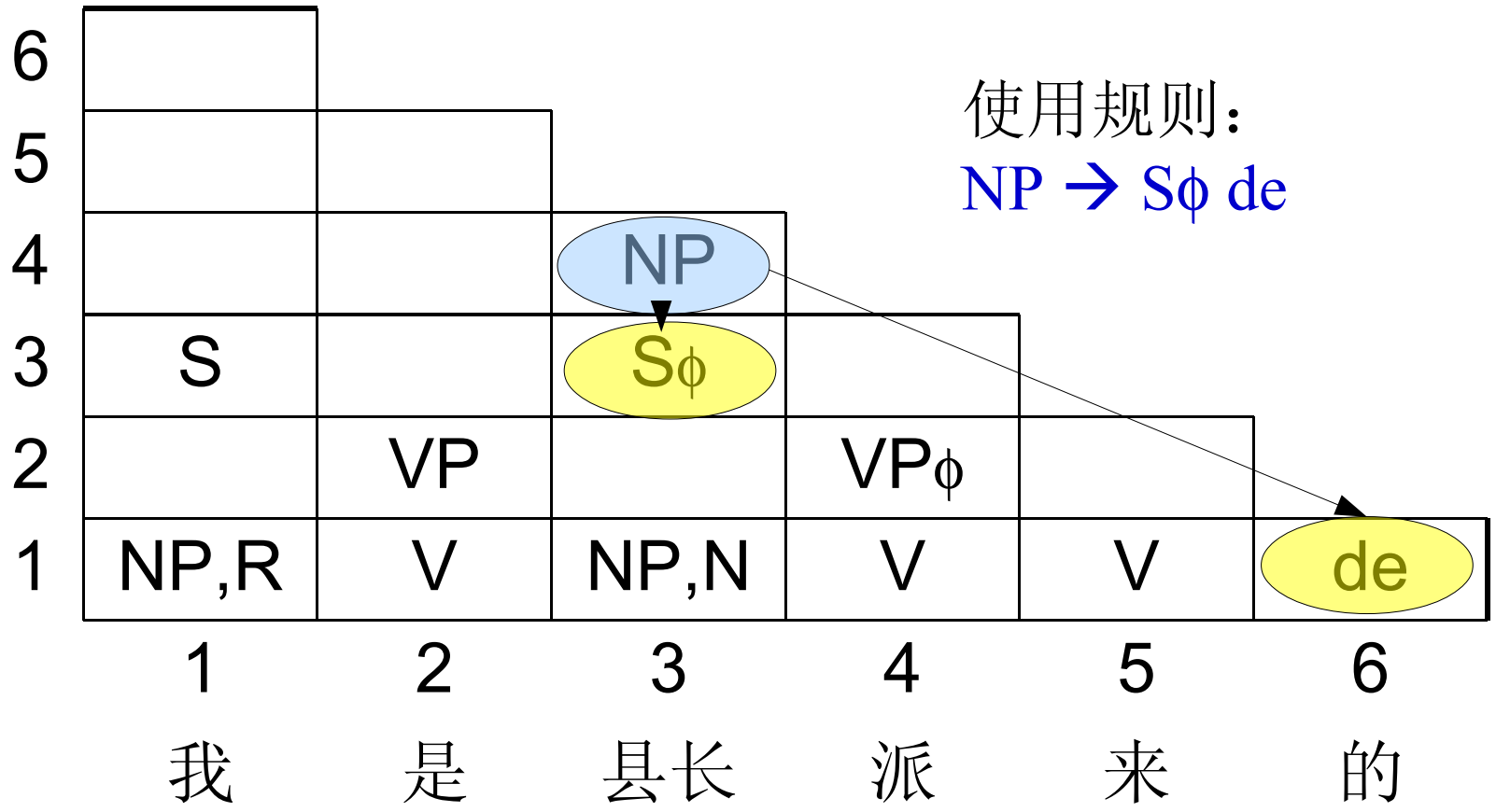


无可用规则！

CYK 算法— Span 的划分 (2)



CYK 算法— Span 的划分 (3)



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。

Earley 算法—概述

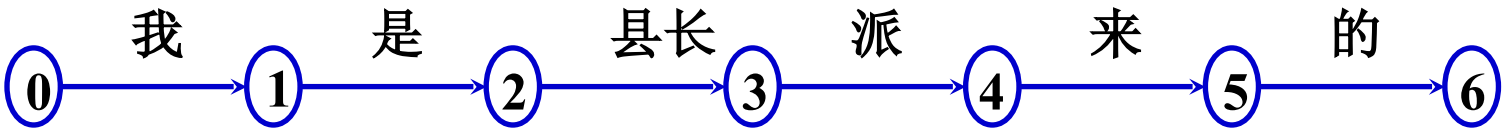
- Earley 算法也是一种并行算法，不需要回溯；
- 类似于 CYK 算法， Earley 算法中也通过一个二维矩阵来存放已经分析过的结果；
- Earley 算法的一个重要贡献是引入了点规则，进一步减少了规则匹配中的冗余操作；
- Earley 算法是一种自顶向下的分析算法。

Earley 算法：点规则

- 所谓点规则，是在规则的右部的终结符或非终结符之间的某一个位置上加上一个圆点，表示规则右部被匹配的程度
- 例子：
  - VP→ · V NP 表示这条规则还没有被匹配
  - VP→ V · NP 表示这条规则右部的 V 已经匹配成功，而 NP 还没有被匹配
  - VP→ V NP · 表示这条已被完全匹配，并形成了一个短语 VP

Earley 算法：数据结构

- 数据结构：二维矩阵 {E(i,j)}
  - 下标 i 和 j 对应于输入句子词语之间的间隔
  - 每个矩阵元素对应于句子中的一个区间（ Span ）



Earley 算法：算法描述

- 初始化：
  - 对于规则集中，所有左端为初始符 S 的规则 S→α ，把 S→ · α 加入到 E(0,0) 中
  - 如果 B→ · A β 在 E(0,0) 中，那么对于所有左端为符号 A 的规则 A→ α ，把 A→ · α 加入到 E(0,0) 中
- 循环执行以下步骤，直到分析成功或失败：
  - 如果 A→α · x_j β 在 E(i,j-1) 中，那么把 A→α x_j · β 加入到 E(i,j) 中
  - 如果 A→ α · B β 在 E(i,j) 中，那么对所有左端为符号 B 的规则 B→γ ，把 B→ · γ 加入到 E(j,j) 中
  - 如果 B→γ 在 E(i,j) 中，且在 E(k,i) 存在 A→ α · B β ，那么把 A→ α B · β 加入到 E(k,j) 中

Earley 算法：数据结构

- 数据结构：二维矩阵 {E(i,j)}，其中每个元素是一个点规则的集合，用来存放句子中单词 i+1 到单词 j 这个跨度上所分析得到的所有点规则
- 还是以“我 是 县长 派 来 的”为例：
  E(0,0)={S→ · NP VP， NP→ · N， NP → · R，
  NP→ · Sϕ de， Sϕ→ · NP VPϕ ……}
  E(0,1)={N→ 我 ·， NP→N ·， S→NP · VP
  …… }
  E(1,1)={VP→ · V V, VP → · V NP …… }
- Earley 算法就是从左到右逐步填充这个二维矩阵的过程

Earley 算法：运行示例

- 初始化：
  - E(0,0)<={S→ · NP VP}
  - E(0,0)<= {NP→ · N, NP → · R, NP→ · Sϕ de}
  - E(0,0)<= {Sϕ→ · NP VPϕ}

Earley 算法：运行示例

- 循环：
  - E(0,1)<={ R→ 我 ·}
  - E(0,1)<={ NP→ R ·}
  - E(0,1)<={S→ NP · VP} {Sϕ→ NP · VPϕ }
  - E(1,1)<={VP → · V NP }
  - E(1,1)<={VPϕ → · V V}
  - ……
  - E(0,6)<={S→ NP VP ·} 成功！

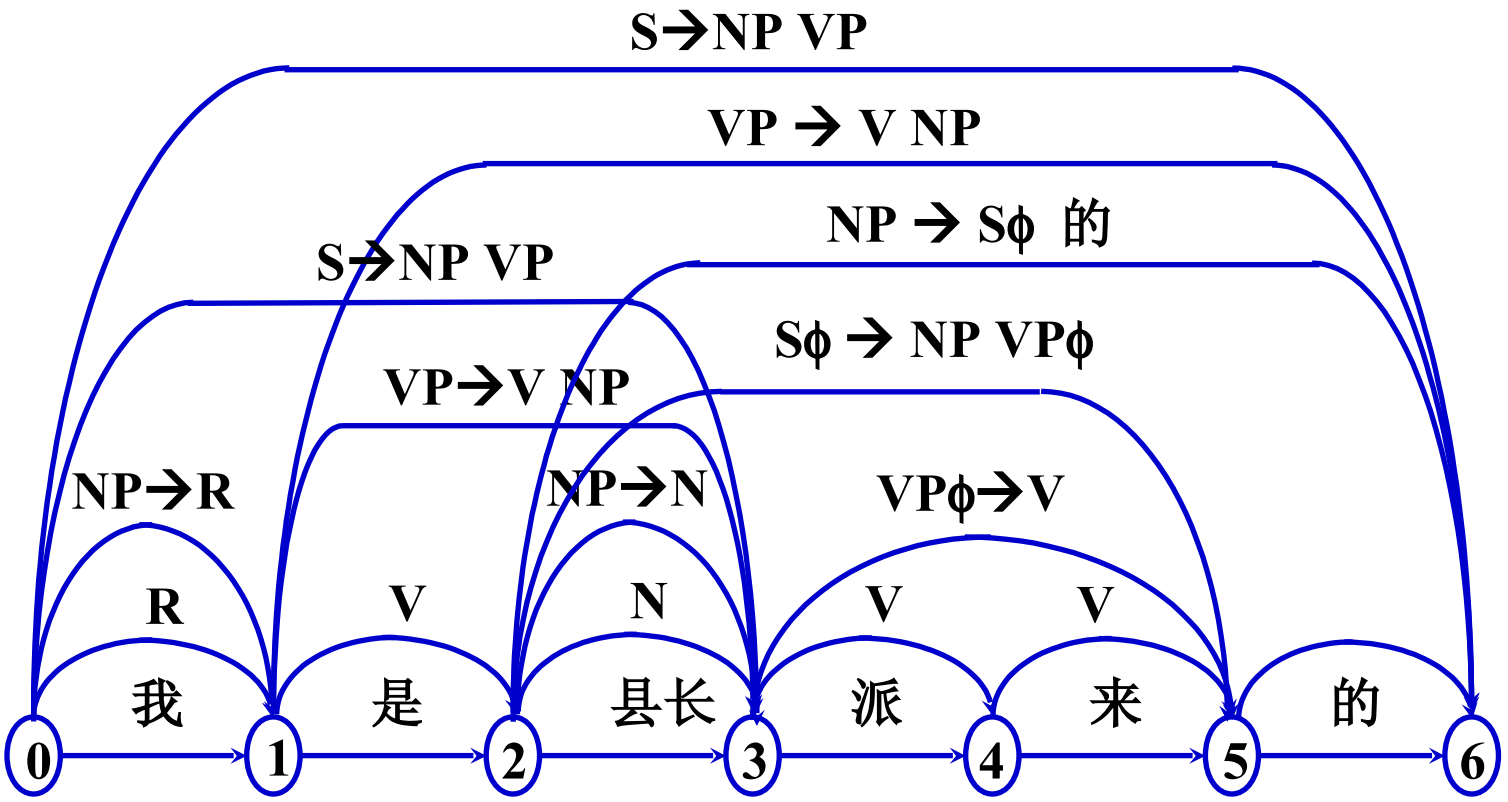
线图（Chart）分析算法

- 线图分析算法 Chart Parsing Algorithm
- 线图分析法的核心是线图（Chart）表示法，线图表示法具有简单、直观的特点；
- 通过修改线图分析法的分析策略，可以方便地模拟很多种分析算法，如自顶向下的分析方法、自底向上的分析方法、左角分析方法等等。

线图表示法 1

- 线图是一个无环有向图（DAG），其中：
  - **结点**：输入句子中词与词之间的每一个间隔为一个结点；结点的标记往往用一个序号来表示；
  - **边（弧）**：对应于句子中的一个短语，边两端的结点给定了短语的边界，边的方向总是从左到右。边上面不仅要标记短语的类型，还需要标记产生该短语的规则。
  - **说明**：在汉语分析中，为了兼容词语切分的歧义，常常将汉字之间的间隔作为一个结点

线图表示法 2


S→NP VP
VP → V NP
NP → Sϕ 的
S→NP VP
Sϕ → NP VPϕ
VP→V NP
NP→R
NP→N
VPϕ→V
R
V
N
V
V
我
是
县长
派
来
的
0
1
2
3
4
5
6

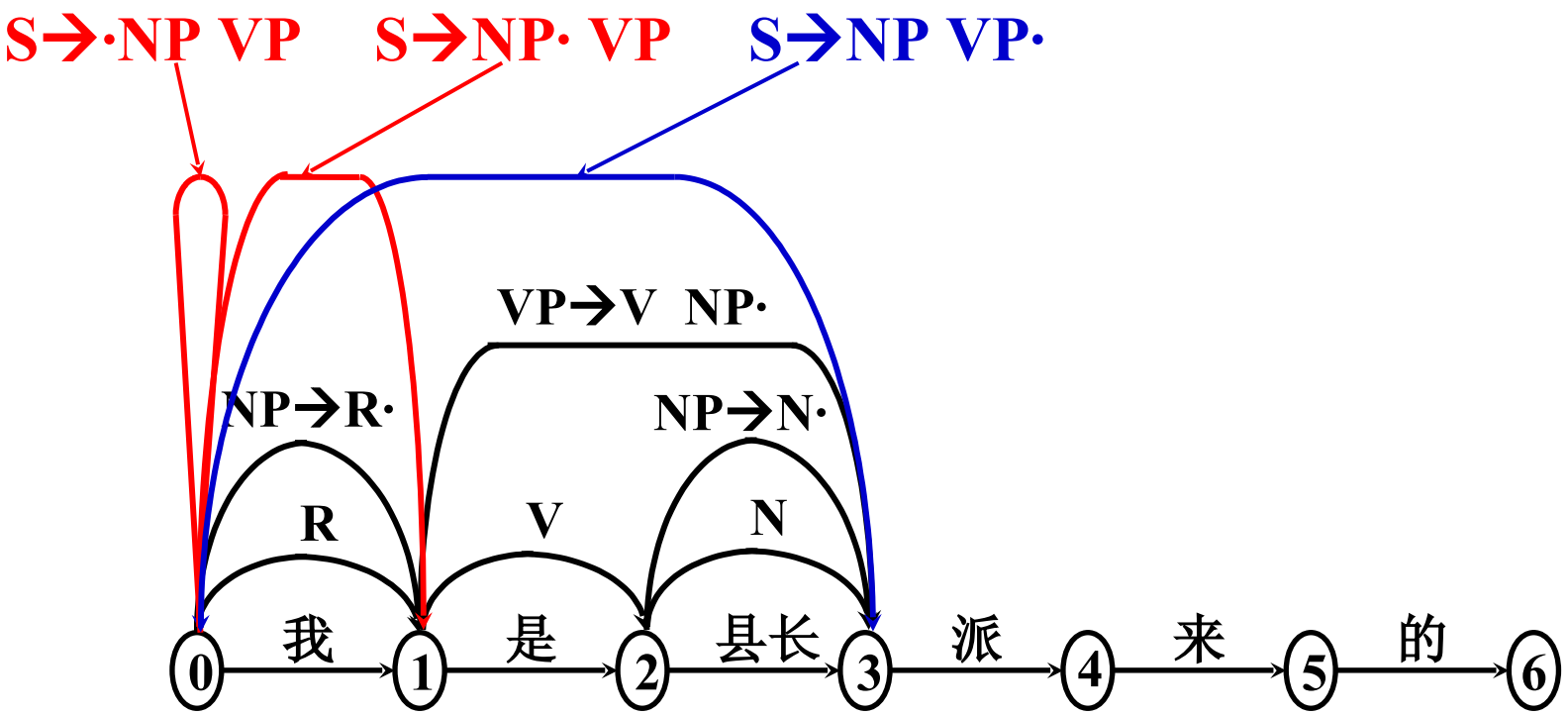
活跃边与非活跃边 1

- 上述记录分析成功的短语的边称为非活跃边。
- 在线图中，还有另一种形式的边，用于记录一条规则不完全分析的结果，称为活跃边，如下所示：

记录方式	边状态	匹配程度	起点	终点	对应词串
<0,0, S→·NP VP>	活跃	**S → ·NP VP**	0	0	
<0,1, S→NP· VP>	活跃	**S → NP· VP**	0	1	我
<0,3, S→NP VP·>	非活跃	**S → NP VP·**	0	3	我是县长

- 活跃边的引入，可以减少规则匹配中的冗余操作，提高句法分析的效率。

活跃边与非活跃边 2



日程表 (Agenda)

- 在线图分析算法中，除了“线图 (Chart)” 以外还有一个重要的数据结构，称为“ 日程表 (Agenda)”
- Chart 分析的过程就是一个不断产生新的边的过程。但是每一条新产生的边并不能立即加入到 Chart 中，而是要作为候选边放到日程表 (Agenda) 中
- 日程表 (Agenda) 实际上是一个边的集合，用于存放已经产生，但是还没有加入到 Chart 中的边。
- 日程表 (Agenda) 中边的排序和存取方式，是 Chart 算法执行策略的一个重要方面

线图分析算法的基本流程

线图分析算法就是一个由日程表驱动的不断循环的过程：

1. 按照**初始化策略**初始化 Agenda
2. 如果 Agenda 为空，那么分析失败
3. 每次按照**日程表组织策略**从 Agenda 中取出一条边
4. 如果取出的边是一条非活跃边，而且覆盖整个句子，那么返回成功
5. 将取出的边加入到 Chart 中，执行**规则匹配策略**和**规则调用策略**，将产生的新边又加入到 Agenda 中
6. 返回第 (2) 步

线图分析算法：初始化策略

- Chart 分析算法开始执行以前，要先将 Agenda 初始化
- 对于不同的句法分析策略，初始化策略也不相同

自底向上分析的规则调用策略

– 将所有单词（含词性）边加入到 Agenda 中。

自顶向下分析的规则调用策略

– 将所有单词（含词性）边加入到 Agenda 中；

– 对于所有形式为 S→W 的规则，产生一条形式为 <0, 0, S→·W> 的边，并加入到 Agenda 中；

线图分析算法：规则匹配策略

- 在 Chart 算法中，边是逐条从 Agenda 中加入到 Chart 中的
- 将每一条边从 Agenda 中取出并加入到 Chart 中时，都要执行以下**规则匹配策略**：
  - 如果新加入一条活跃边形式为： <i, j, A→W1· B W2> ，那么对于 Chart 中所有形式为： <j, k, B→W3> 的非活跃边，生成一条形式为 <i, k, A→W1 B· W2> 的新边，并加入到 Agenda 中；
  - 如果新加入一条非活跃边形式为： <j, k, B→W3> ，那么对于 Chart 中所有形式为： <i, j, A→W1· B W2> 的活跃边，生成一条形式为 <i, k, A→W1 B· W2> 的新边，并加入到 Agenda

上面 A 、 B 为非终结符， W1 、 W2 、 W3 为终结符和非终结符组成的串，其中 W1 、 W2 允许为空， W3 不允许为空

线图分析算法：规则调用策略

- 对于不同的句法分析策略，规则调用策略也不相同：

  自底向上分析的规则调用策略

  – 如果要加入一条形式为 <i, j, C→W1·> 的边到 Chart 中，那么对于所有形式为 B→C W2 的规则，产生一条形式为 <i, i, B→· C W2> 的边加入到 Agenda 中

  自顶向下分析的规则调用策略

  – 如果要加入一条形式为 <i, j, C→W1·B W2> 的边到 Chart 中，那么对于所有形式为 B→W 的规则，产生一条形式为 <j, j, B→·W> 的边，并加入到 Agenda 中

线图分析算法：日程表组织策略

- 通过日程表组织的不同策略，可以分别实现深度优先和广度优先等搜索方法

  深度优先的日程表组织策略

  – 将日程表按照堆栈的形式，每次从日程表中取出最后加入的结点；

  广度优先的日程表组织策略

  – 将日程表按照队列的形式，每次从日程表中取出最早加入的结点；

线图分析算法：细节处理

- 前面的讨论中忽略了两个细节，在实现一个系统时应该考虑到：
  - 考虑到可能通过多种途径生成一条完全相同的边，所以每次从 Agenda 中取出一条新边加入 Chart 时，要先检查一下 Chart 中是否已经有相同的边，如果有，那么删除这条边，直接进入下一个循环
  - 为了生成最后的句法结构树，每一条边中还应该记录该条边的子句法成分所对应的边

线图算法分析示例（1）

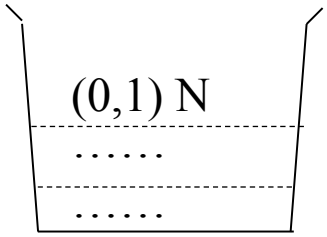
此例子来自詹卫东《计算语言学概论》讲义，
在此表示感谢！

chart

active arc

⓪ ① ② ③ ④ ⑤ ⑥

agenda

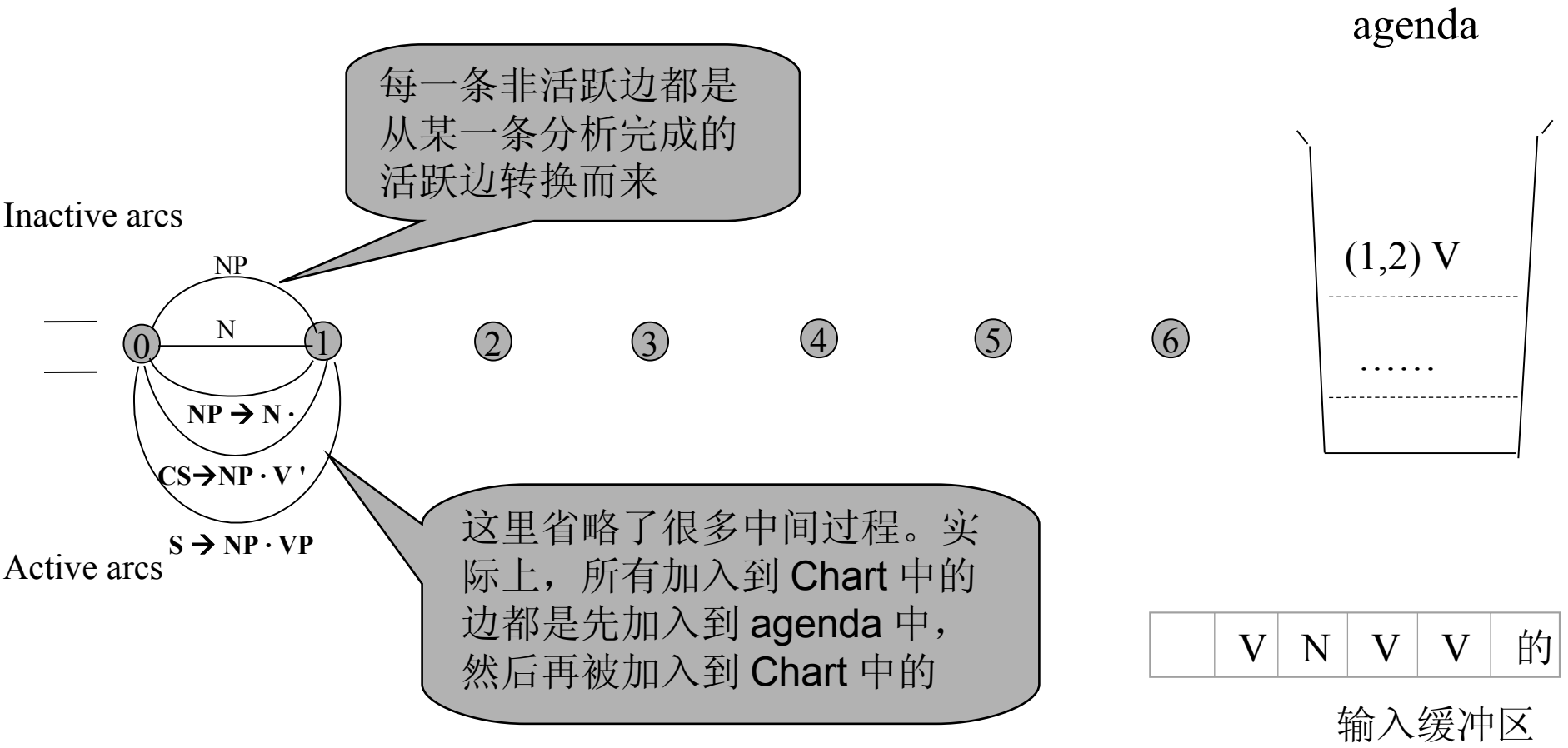


输入缓冲区

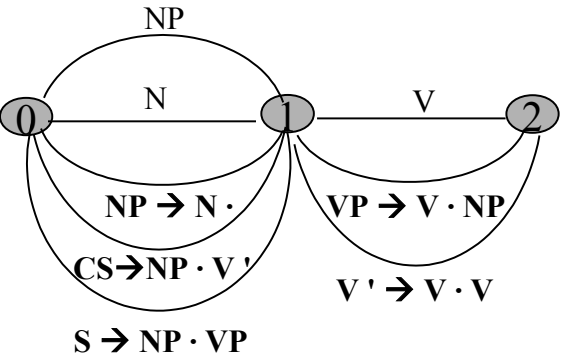
N	V	N	V	V	的

我 是 县长 派 来 的

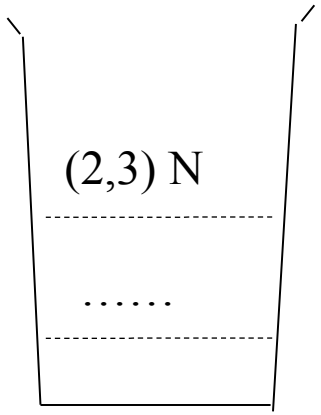
线图算法分析示例（２）


agenda
每一条非活跃边都是从某一条分析完成的活跃边转换而来
Inactive arcs
NP
N
0
1
2
3
4
5
6
NP → N ·
CS→NP · V '
S → NP · VP
Active arcs
(1,2) V
……
这里省略了很多中间过程。实际上，所有加入到 Chart 中的边都是先加入到 agenda 中，然后再被加入到 Chart 中的
V
N
V
V
的
输入缓冲区

线图算法分析示例（3）



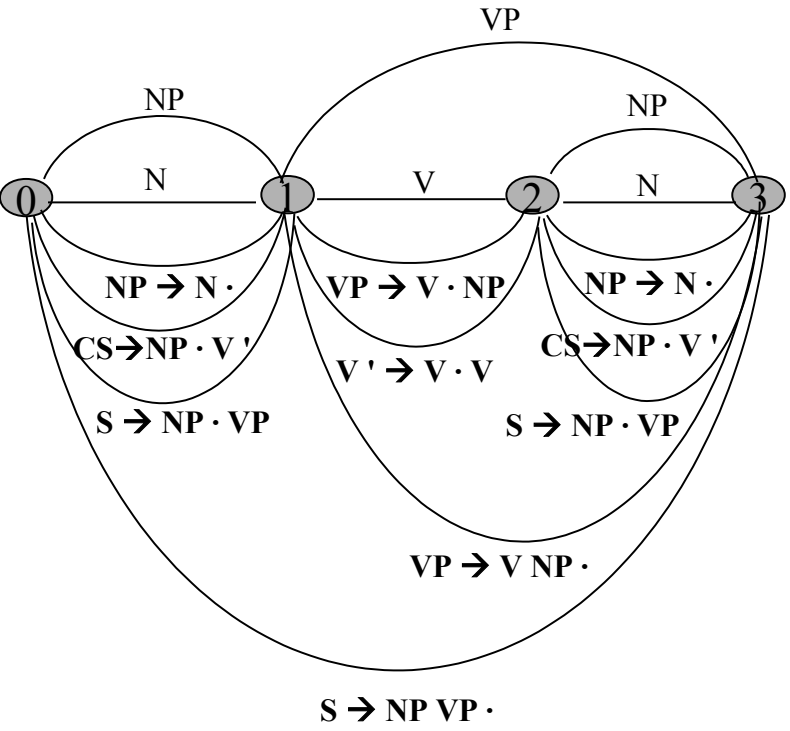
agenda



		N	V	V	的

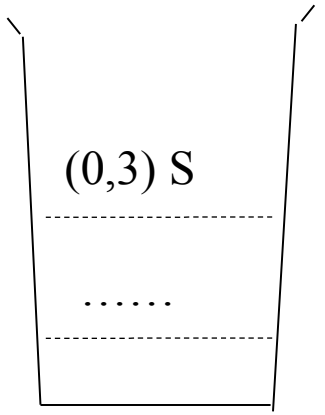
输入缓冲区

线图算法分析示例（4）



4 5 6

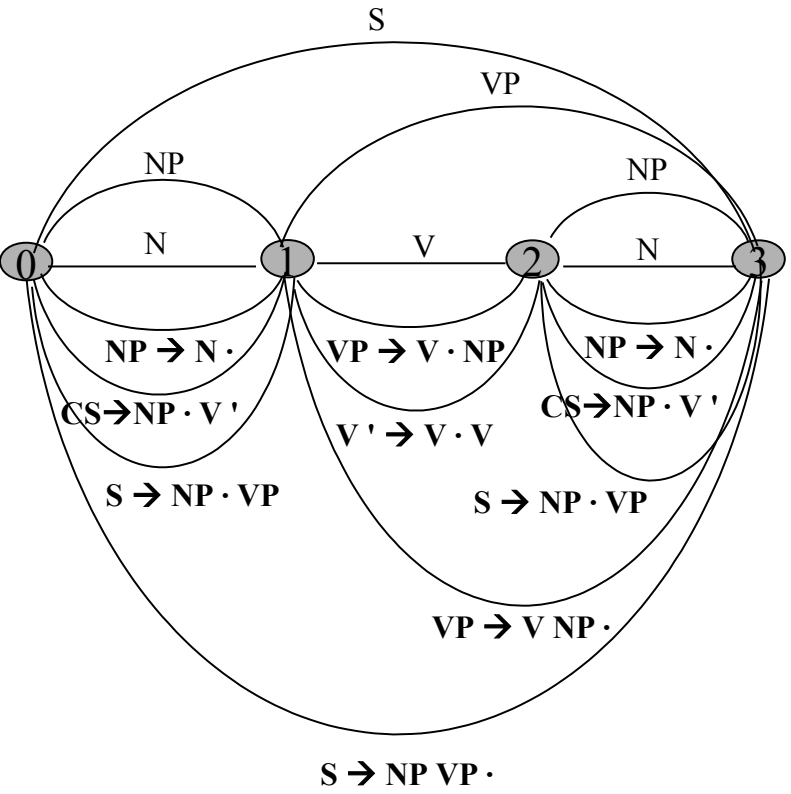
agenda



			V	V	的

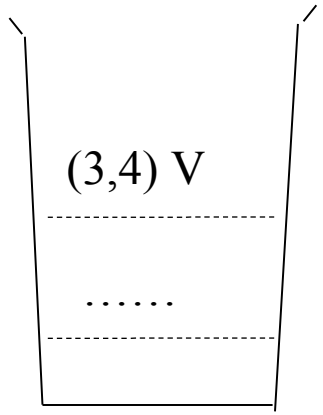
输入缓冲区

线图算法分析示例（5）



4 5 6

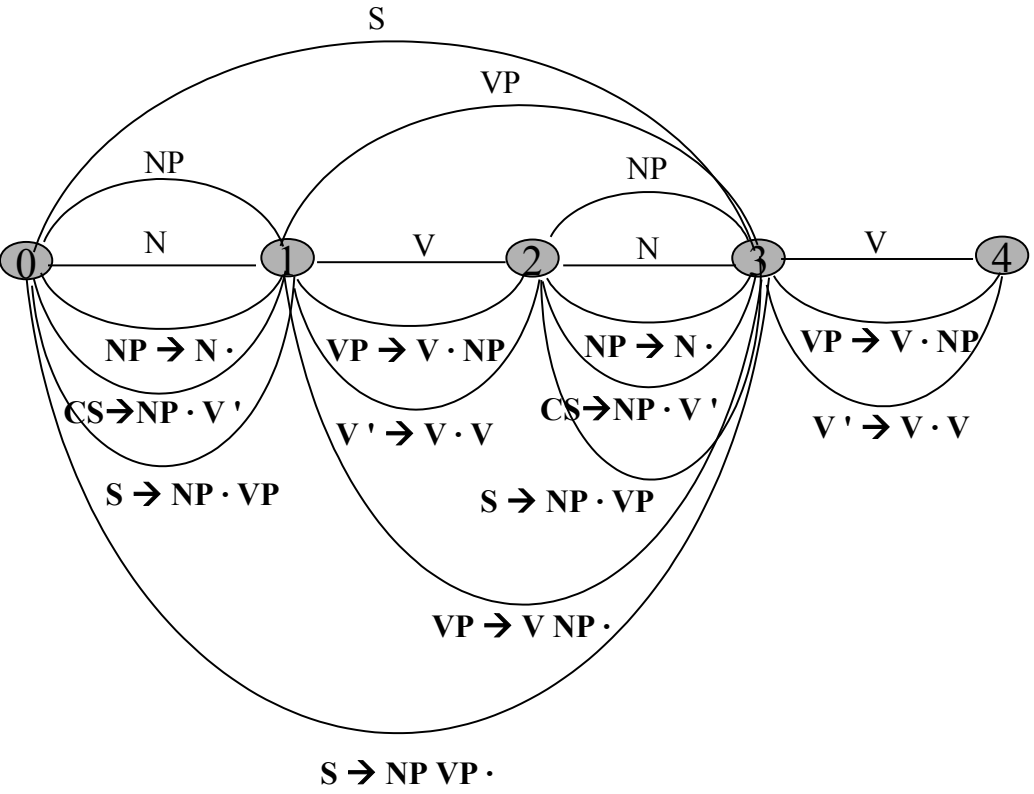
agenda



			V	V	的

输入缓冲区

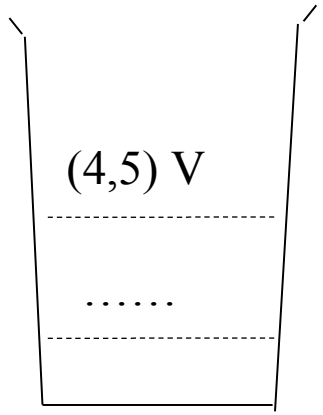
线图算法分析示例（6）



5

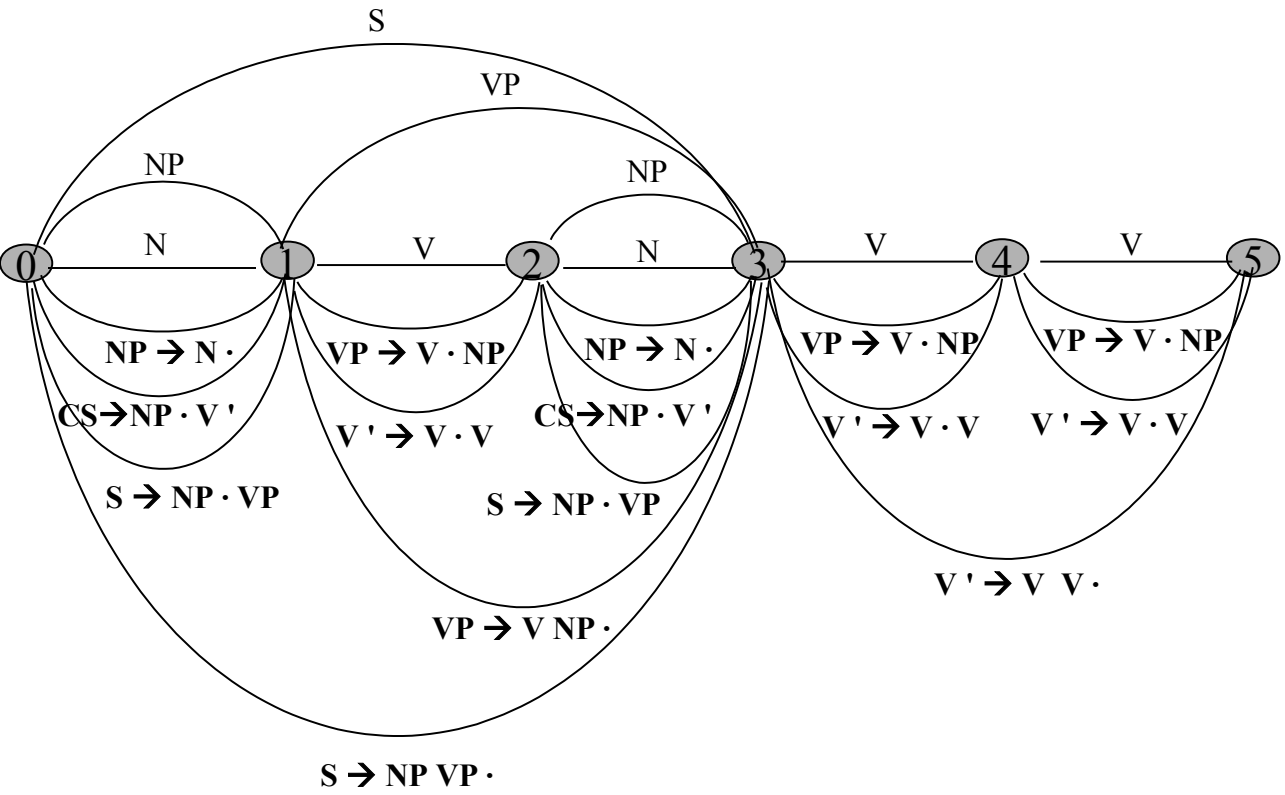
6

agenda

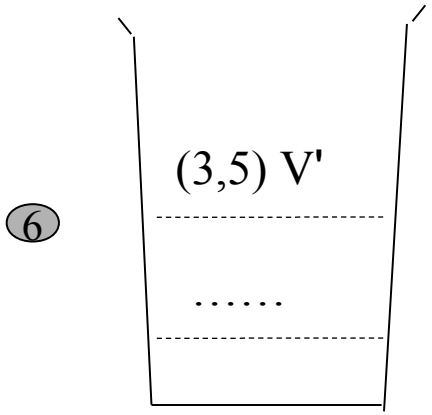


输入缓冲区

线图算法分析示例（7）

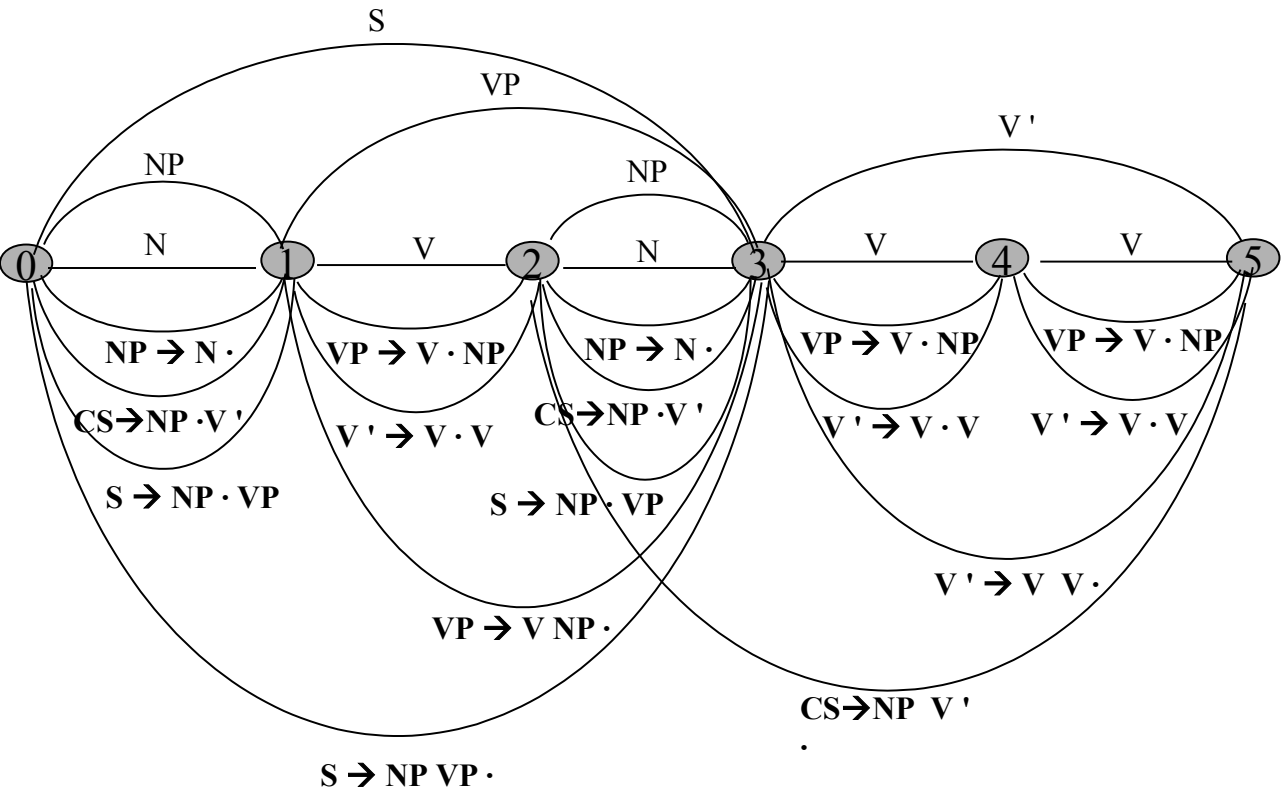


agenda

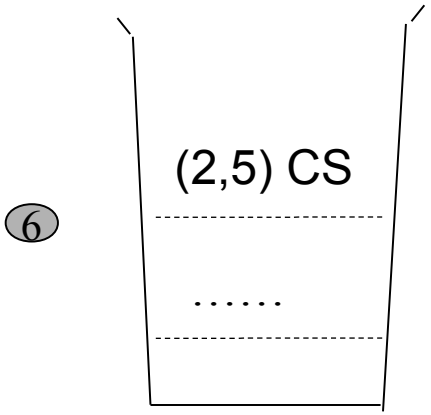


输入缓冲区

线图算法分析示例（8）

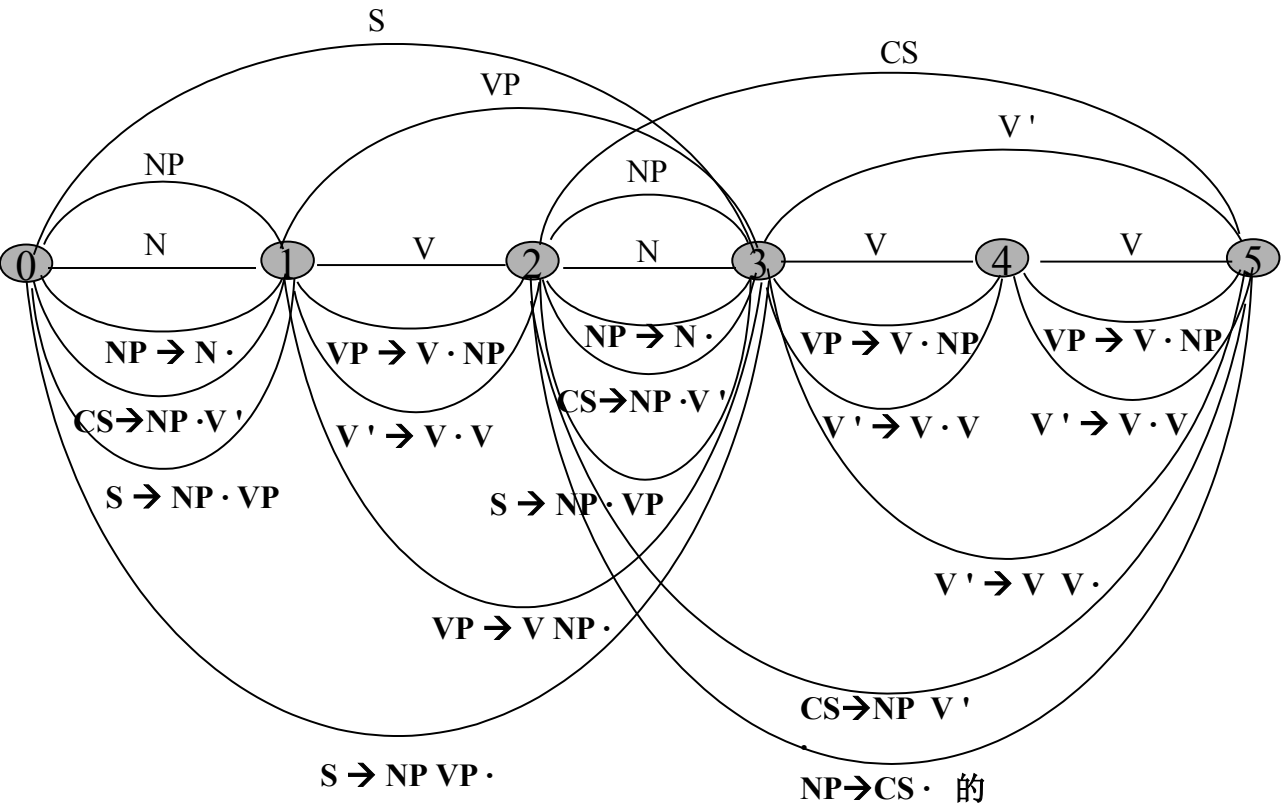


agenda

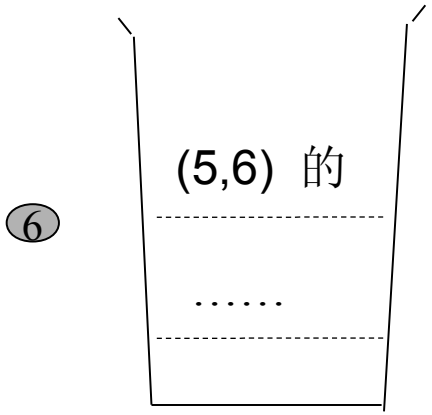


输入缓冲区

线图算法分析示例（9）

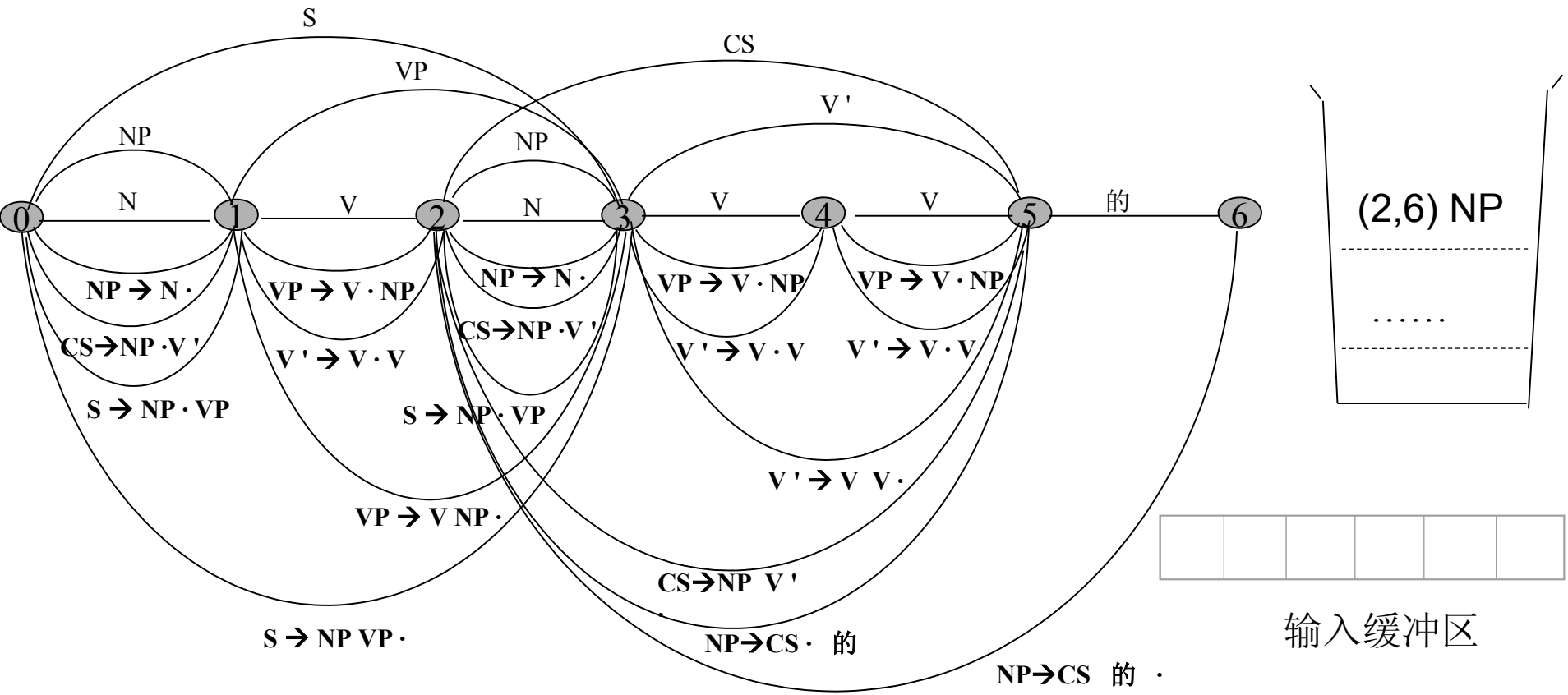


agenda

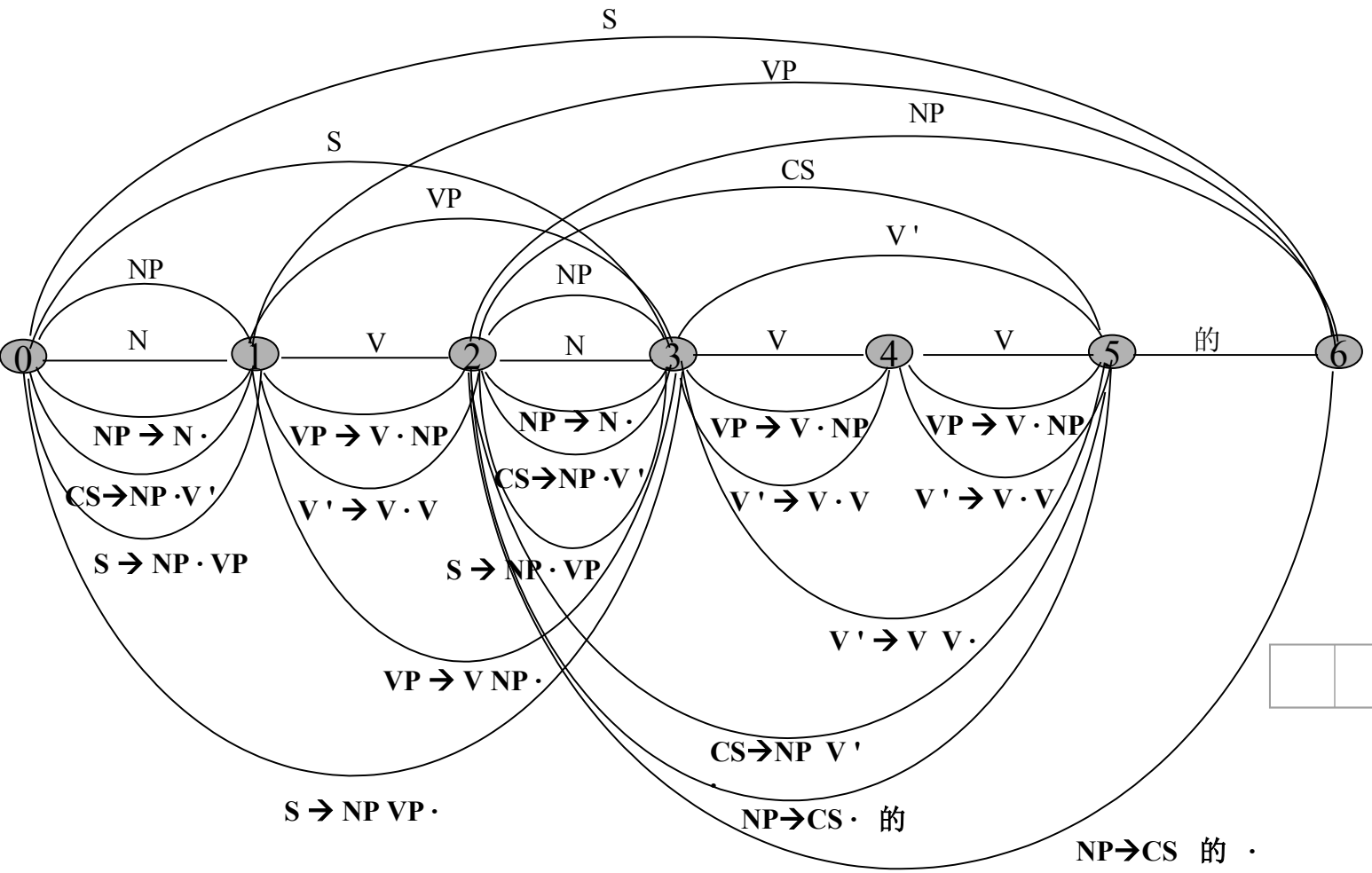


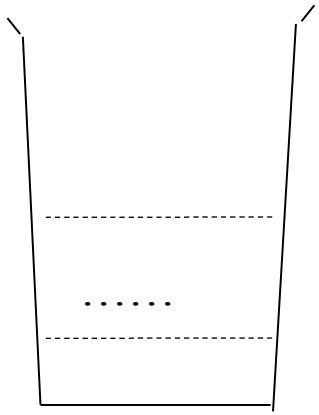
输入缓冲区

线图算法分析示例（10）


agenda
S
VP
NP
N
CS
V '
NP
V
N
V
V
的
0
1
2
3
4
5
6
NP → N ·
CS→NP ·V '
S → NP · VP
VP → V · NP
V ' → V · V
NP → N ·
CS→NP ·V '
S → NP · VP
VP → V NP ·
VP → V · NP
V ' → V · V
V ' → V V ·
VP → V · NP
V ' → V · V
CS→NP V '
NP→CS · 的
S → NP VP ·
NP→CS 的 ·
(2,6) NP
.....
输入缓冲区

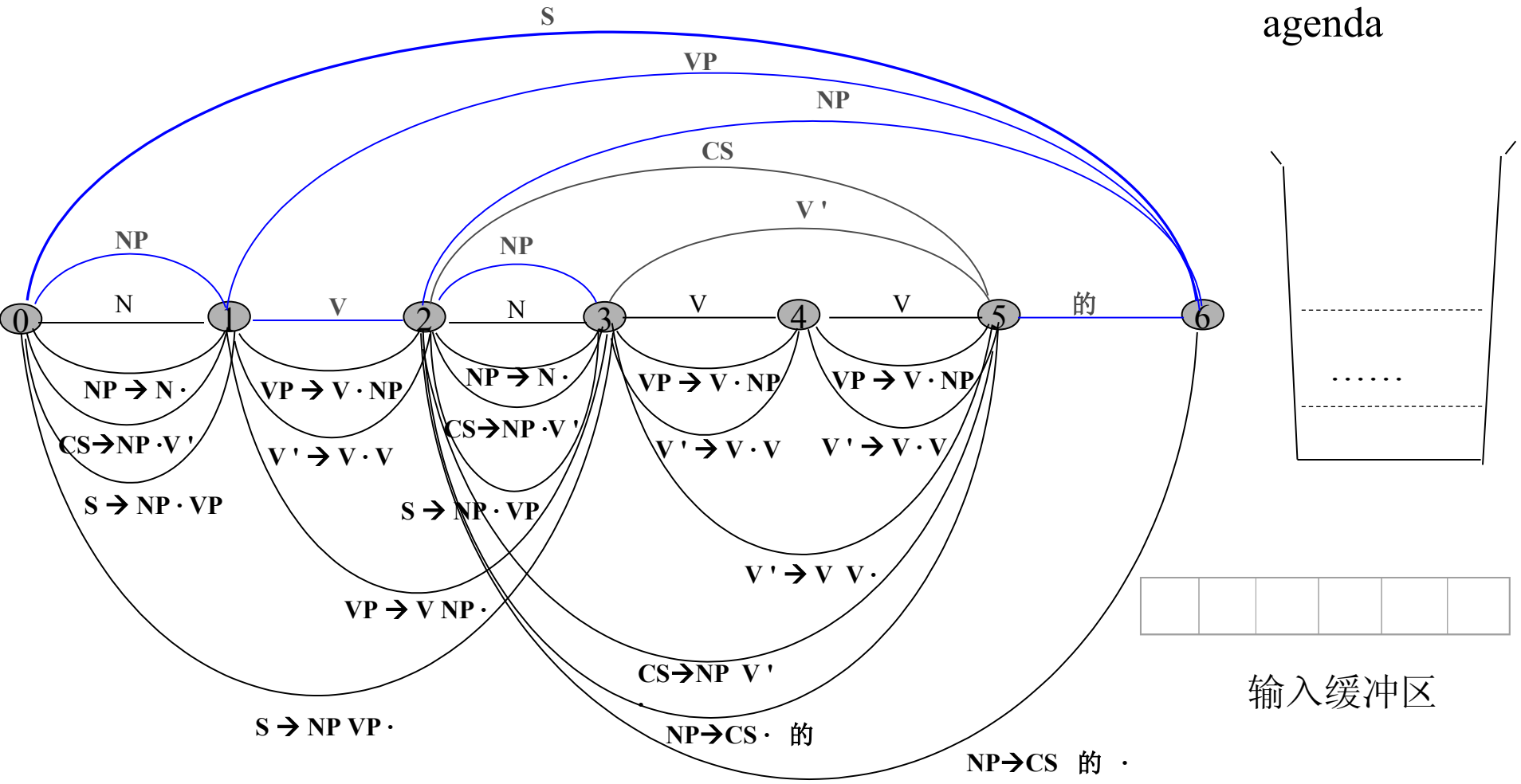
线图算法分析示例（11）





输入缓冲区

线图算法分析示例（12）



agenda

输入缓冲区

线图分析算法：总结

- Chart 算法具有直观和灵活的特点；
- 通过修改分析过程中的一些具体策略， Chart 分析算法可以模拟很多种其他句法分析算法。
- 白硕和张浩在“角色反演算法”（软件学报）中，把 Tomita 算法中“向前看（ look ahead ）”的思想结合到 Chart 分析算法中，提出了一种“角色反演算法”，可以减少 Chart 分析算法中垃圾边的数量而又不影响最后的分析结果，提高分析的效率。

Tomita 算法 (GLR 算法) 概述

- Tomita 算法又称为 Generalized LR 算法、 GLR 算法、富田算法、富田胜算法
- LR 实际上就是一种确定性的移进—归约算法，通过一个分析表来确定当前需要采取的动作
- Tomita 算法是对 LR 算法的改进，使得处理形式语言用的 LR 算法也能够处理自然语言
- Tomita 算法的优点是效率非常高
- Tomita 算法的缺点是构造分析表需要花费较长时间

LR 分析表示例（原始形式）

下一个符号

栈顶符号

状态	动作表（Action）				转移表（Go to）							
	N	V	的	$	N	V	的	S	NP	VP	CS	V'
0	移进				1			3	2		4	
1		归约 2										
2		移进				6				5		7
3				成功								
4			移进				8					
5				归约 1								
6	移进	移进			10	9			11		12	
7			归约 5									
8		归约 3										
9			归约 6									
10		归约 2		归约 2								
11		移进		归约 4		13						7
12			移进				14					
13		移进				9						
14		归约 3		归约 3								

动作表归约后面的数字表示规则序号，转移表中的数字表示转移到的状态。

LR 分析表示例（简化形式）

状态	动作表（Action）				转移表（Go to）				
	N	V	的	$	S	NP	VP	CS	V'
0	移进 1				3	2		4	
1		归约 2							
2		移进 6					5		7
3				成功					
4			移进 8						
5				归约 1					
6	移进 10	移进 9				11		12	
7			归约 5						
8		归约 3							
9			归约 6						
10		归约 2		归约 2					
11		移进 13		归约 4					7
12			移进 14						
13		移进 9							
14		归约 3		归约 3					

Tomita 算法的基本思想

- GLR 分析表允许有多重入口（即一个格子里有多个动作）
- 将线性分析栈改进为图分析栈处理分析动作的歧义（分叉）
- 采用共享子树结构来表示局部分析结果，节省空间开销
- 通过节点合并，压缩局部歧义

GLR 分析示例

I	saw	a	girl	with	a	telescope
Pron	V	Det	N	Prep	Det	N

此例子来自詹卫东《计算语言学概论》讲义，在此表示感谢！

(0) $S' \rightarrow S$

(1) $S \rightarrow NP\ VP$

(2) $VP \rightarrow V$

(3) $VP \rightarrow V\ NP$

(4) $VP \rightarrow V\ NP\ NP$

(5) $VP \rightarrow VP\ PP$

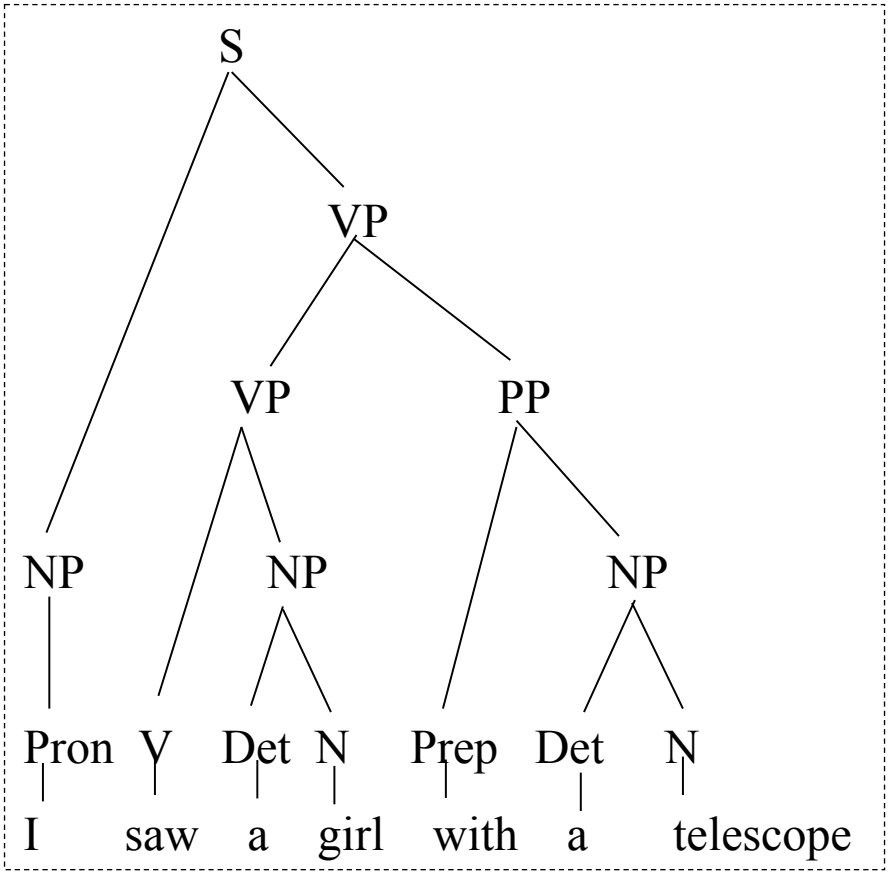
(6) $NP \rightarrow Det\ N$

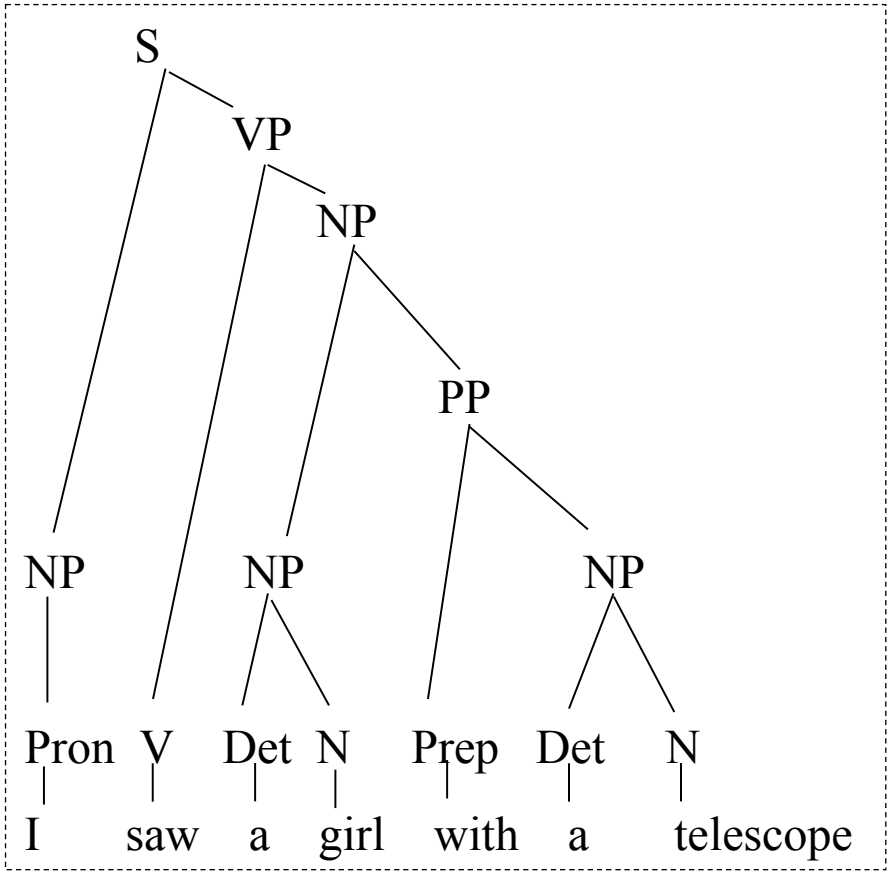
(7) $NP \rightarrow Pron$

(8) $NP \rightarrow NP\ PP$

(9) $PP \rightarrow Prep\ NP$

两棵句法树

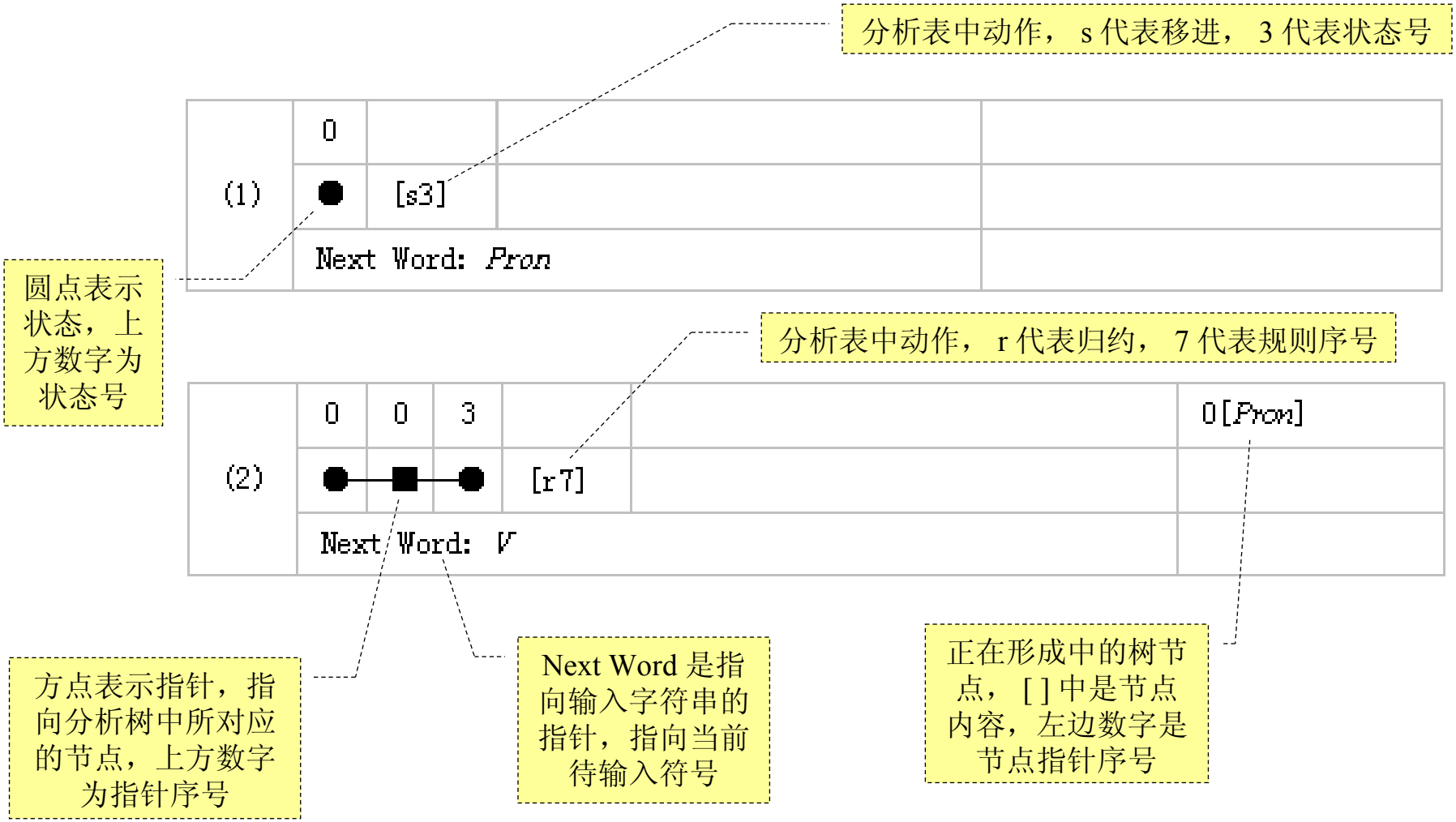

S
VP
VP
PP
NP
NP
NP
Pron
V
Det
N
Prep
Det
N
I
saw
a
girl
with
a
telescope


S
VP
NP
PP
NP
NP
NP
Pron
V
Det
N
Prep
Det
N
I
saw
a
girl
with
a
telescope

GLR 分析表

状态	ACTION						GOTO			
	Det	*N*	*Prep*	*Pron*	*V*	$	*NP*	*PP*	S	VP
0	s2			s3			1		4	
1			s8		s6			7		5
2		s10								
3	r7		r7	r7	r7	r7	1			
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		

GLR 分析过程（1）


分析表中动作， s 代表移进， 3 代表状态号
(1)
0
[s3]
Next Word: Pron
圆点表示状态，上方数字为状态号
分析表中动作， r 代表归约， 7 代表规则序号
(2)
0 0 3
[r7]
0[Pron]
Next Word: V
方点表示指针，指向分析树中所对应的节点，上方数字为指针序号
Next Word 是指向输入字符串的指针，指向当前待输入符号
正在形成中的树节点， [ ] 中是节点内容，左边数字是节点指针序号

GLR 分析过程（2）

	0	1	1			0[*Pron*]
(3)	●	■	●	[s6]		1[*NP*(0)]
	Next Word: *V*					

	0	1	1	2	6			0[*Pron*]
(4)	●	■	●	■	●	[s2]		1[*NP*(0)]
	Next Word: *Det*							2[*V*]

	0	1	1	2	6	3	2			0[*Pron*]
(5)	●	■	●	■	●	■	●	[s10]		1[*NP*(0)]
	Next Word: *N*									2[*V*]
										3[*Det*]

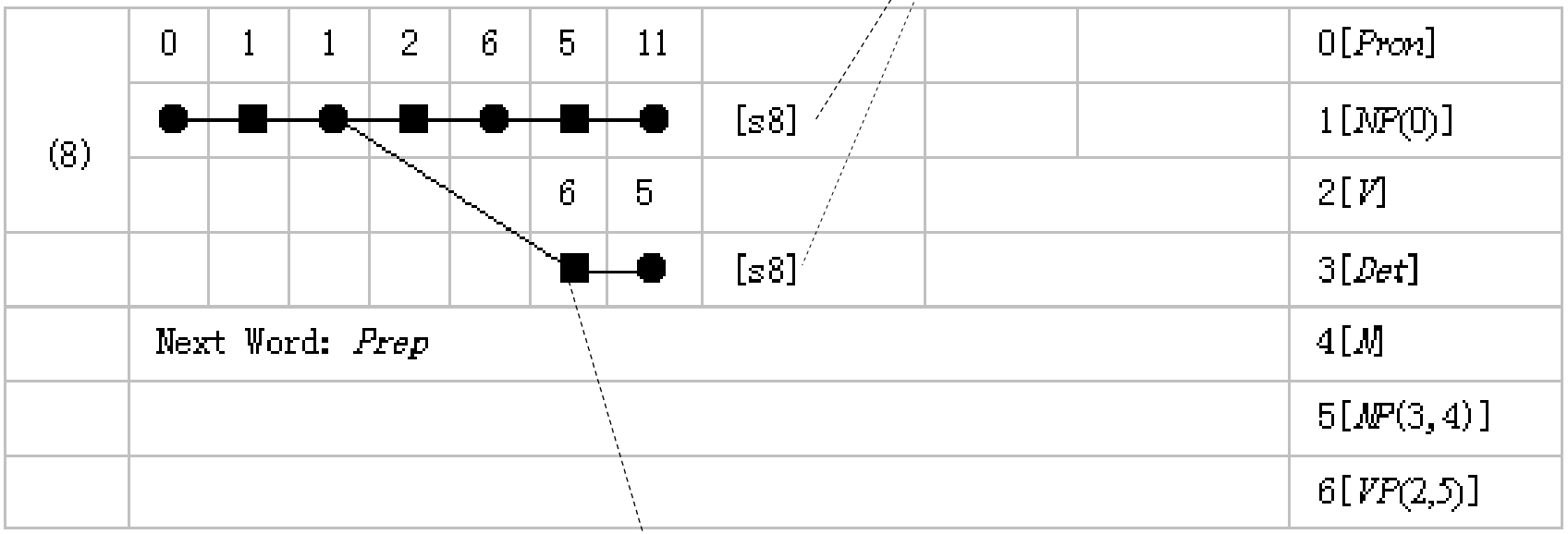
GLR 分析过程（3）

	0	1	1	2	6	3	2	4	10			0[*Pron*]
(6)										[r6]		1[*NP*(0)]
	Next Word: *Prep*											2[*V*]
												3[*Det*]
												4[*M*]

	0	1	1	2	6	5	11					0[*Pron*]
(7)								[s8][r3]				1[*NP*(0)]
	Next Word: *Prep*											2[*V*]
												3[*Det*]
												4[*M*]
												5[*NP*(3,4)]

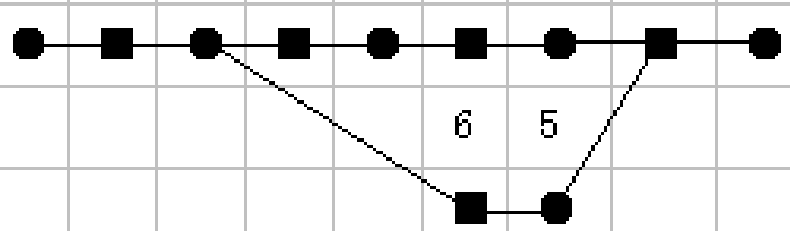
GLR 分析过程（4）

相同的栈顶，相同的 Next Word，
执行相同的 shift 操作后可以分支合并



执行 r3，图分析栈裂变为两个栈顶
每个 reduce 操作产生一个分支

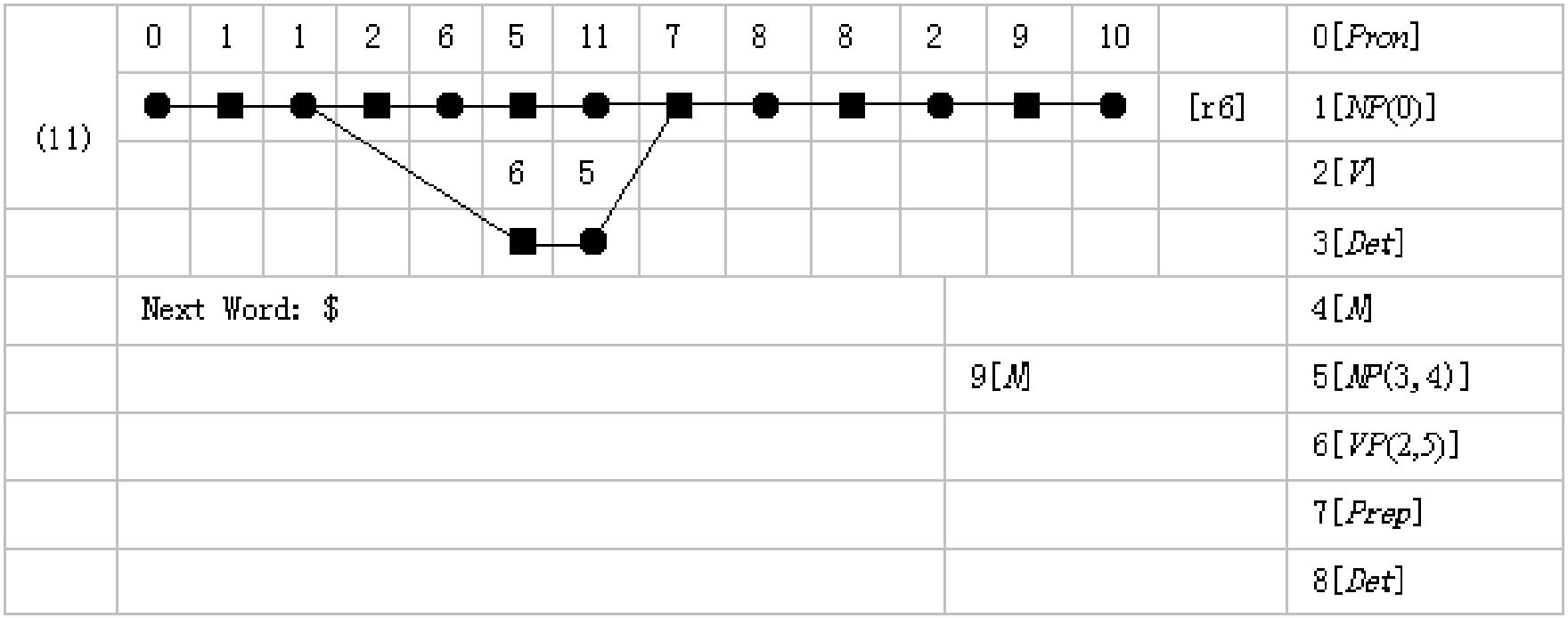
GLR 分析过程（5）

(9)	0	1	1	2	6	5	11	7	8			0[*Pron*]
										[s2]		1[*NP*(0)]
						6	5					2[*V*]
												3[*Det*]
	Next Word: *Det*											4[*M*]
												5[*NP*(3,4)]
												6[*VP*(2,5)]
												7[*Prep*]

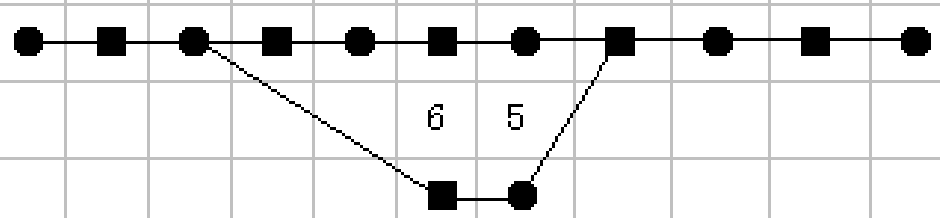
GLR 分析过程（6）

	0	1	1	2	6	5	11	7	8	8	2	0[*Pron*]
(10)											[s10]	1[*NP*(0)]
						6	5					2[*V*]
												3[*Det*]
	Next Word: *N*											4[*N*]
												5[*NP*(3,4)]
												6[*VP*(2,5)]
												7[*Prep*]
												8[*Det*]

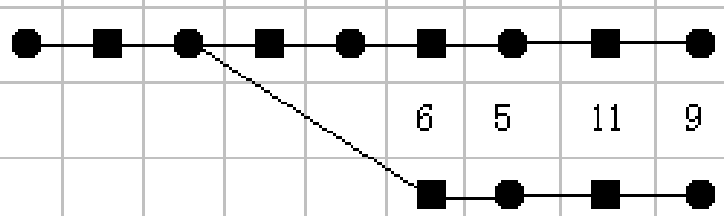
GLR 分析过程（7）


(11)
0 1 1 2 6 5 11 7 8 8 2 9 10
[r6]
6 5
Next Word: $
0[Pron]
1[NP(0)]
2[V]
3[Det]
4[M]
9[M]
5[NP(3,4)]
6[VP(2,5)]
7[Prep]
8[Det]

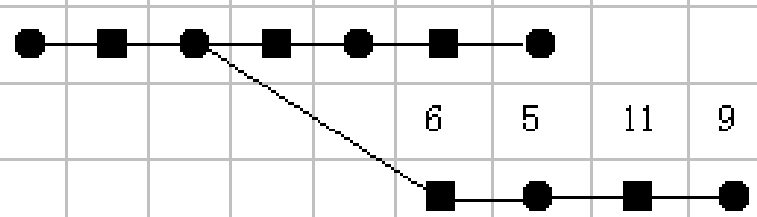
GLR 分析过程（8）

	0	1	1	2	6	5	11	7	8	10	13			0[*Pron*]
(12)												[r9]		1[*NP*(0)]
						6	5							2[*V*]
														3[*Det*]
	Next Word: $													4[*M*]
												9[*M*]		5[*NP*(3,4)]
												10[*NP*(8,9)]		6[*VP*(2,5)]
														7[*Prep*]
														8[*Det*]

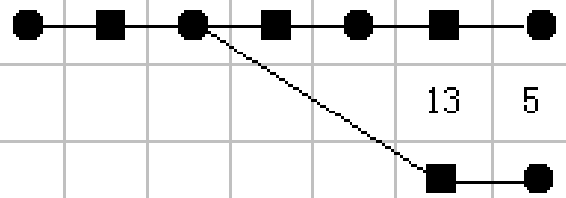
GLR 分析过程（9）

	0	1	1	2	6	5	11	11	7						0[*Pron*]
(13)										[r8]					1[*NP*(0)]
						6	5	11	9						2[*V*]
										[r5]					3[*Det*]
	Next Word: $														4[*N*]
												9[*N*]			5[*NP*(3,4)]
												10[*NP*(8,9)]			6[*VP*(2,5)]
												11[*PP*(7,10)]			7[*Prep*]
															8[*Det*]

GLR 分析过程（ 10 ）

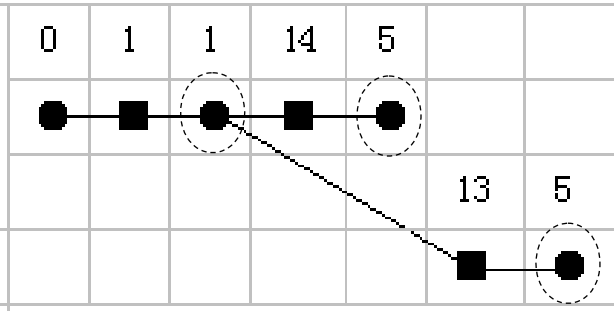
	0	1	1	2	6	12	11								0[*Pron*]
(14)										[r3]					1[*NP*(0)]
						6	5	11	9						2[*V*]
										[r5]					3[*Det*]
	Next Word: $														4[*M*]
												9[*M*]			5[*NP*(3,4)]
												10[*NP*(8,9)]			6[*VP*(2,5)]
												11[*PP*(7,10)]			7[*Prep*]
												12[*NP*(5,11)]			8[*Det*]

GLR 分析过程（ 11 ）

(15)	0	1	1	2	6	12	11							0[*Pron*]
								[r3]						1[*NP*(0)]
						13	5							2[*V*]
								[r1]						3[*Det*]
	Next Word: $													4[*M*]
											10[*NP*(8,9)]			5[*NP*(3,4)]
											11[*PP*(7,10)]			6[*VP*(2,5)]
											12[*NP*(5,11)]			7[*Prep*]
											13[*VP*(6,11)]			8[*Det*]
														9[*M*]

GLR 分析过程（12）

(16)



[r1]

[r1]

Next Word: $

0[*Pron*]

1[*NP*(0)]

2[*V*]

3[*Det*]

4[*N*]

12[*NP*(5,11)]

13[*VP*(6,11)]

14[*VP*((2,12))

5[*NP*(3,4)]

6[*VP*(2,5)]

7[*Prep*]

8[*Det*]

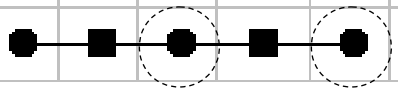
9[*N*]

10[*NP*(8,9)]

11[*PP*(7,10)]

相同标记相同跨度，
结点可以合并

GLR 分析过程（ 13 ）

	0	1	1	13	5										0[*Pron*]
(17)						[r1]									1[*NP*(0)]
	Next Word: $														2[*V*]
											9[*M*]				3[*Det*]
											10[*NP*(8,9)]				4[*M*]
											11[*PP*(7,10)]				5[*NP*(3,4)]
											12[*NP*(5,11)]				6[*VP*(2,5)]
											13[*VP*(6,11)(2,12)]				7[*Prep*]
															8[*Det*]

GLR 分析过程（ 14 ）

	0	14	4												0[*Pron*]
(18)	●—■—●			[acc]											1[*NP*(0)]
	Next Word: $														2[*V*]
													9[*M*]		3[*Det*]
													10[*NP*(8,9)]		4[*M*]
													11[*PP*(7,10)]		5[*NP*(3,4)]
													12[*NP*(5,11)]		6[*VP*(2,5)]
													13[*VP*(6,11)(2,12)]		7[*Prep*]
													14[*S*(1,13)]		8[*Det*]

GLR 分析树（压缩—共享森林）