基于上下文无关文法的句法分析

常宝宝
北京大学计算语言学研究所
chbb@pku.edu.cn

句法分析导引

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

句法学(syntax)

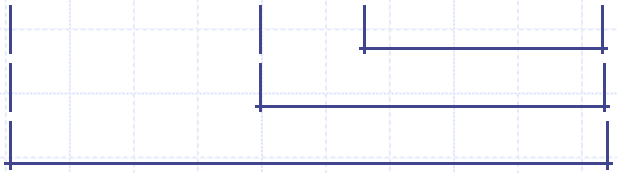
*In linguistics, **syntax** is the study of the rules, or "patterned relations", that govern the way the words in a sentence come together. It concerns how different words are combined into clauses, which, in turn, are combined into sentences.*

From Wikipedia, the free encyclopedia

语言学中研究句子组成规则的学科是句法学

句子成分分析

- 句子是词的线性序列，但词和词之间结合的松紧程度并不一样。
  - 今天 上午 我们 有 两 节 课
- 句子在构造上具有层次性，较小的成分还可以进一步组成较大的成分。
  - 今天 上午 我们 有 两 节 课



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

英语中的短语举例

- 名词性短语(NP)
  the flight　the flight from Beijing to Shanghai
  all the flights　two flights　a nonstop flight
  any flights arriving after eleven a.m.
- 动词性短语(VP)
  arrive on Sunday　show me that book
  buy two books　prefer to go by Airchina
- 介词短语(PP)
  from Beijing　with a telescope　from Beijing to Shanghai
- 形容词性短语
  very easy　least expensive
- ……

汉语中的短语举例

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

句法知识的形式化

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

CFG的形式定义

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

如何用CFG描写英语句法规则？

- 名词性短语

NP → Det Nominal	*a flight the flight*
NP → ProperNoun	*Beijing Chomsky*
NP → Pronoun	*he they him*
Nominal → Noun | Noun Nominal	*flight coffee bean*
NP → NP PP	*all flights from Beijing*
……	

- 动词性短语

VP → Verb	*disappear*
VP → Verb NP	*eat a sandwich*
VP → Verb NP NP	*show me the book*
VP → VP PP	*see a girl with a telescope*
……	

如何用CFG描写汉语句法规则？

- 名词性短语

np → *n* | *r*	桌子、他们
np → *mp np*	一本书、五斤牛肉
np → *ap np*	英俊的小伙子
np → *np np*	英语教师
np → *vp u*	卖菜的
……	

- 动词性短语

vp → *v*	死
vp → *vp np*	吃苹果
vp → *dp vp*	认真准备
vp → *vp ap*	洗干净
……	

一个简单的英语语法

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$

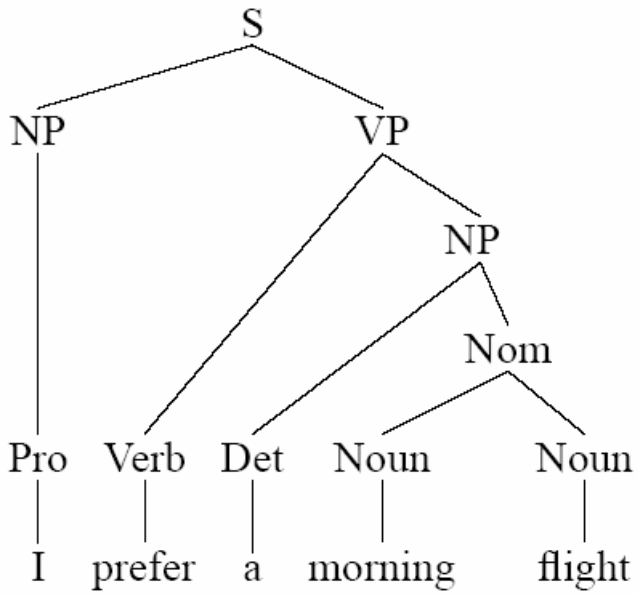
上下文无关文法

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

句法分析

- 句法分析的任务
  - 对给定的自然语言句子，分析并得到其句法结构。
  - 例如：I prefer a morning flight

- 句子的句法结构通常表示为句法树(parse)。

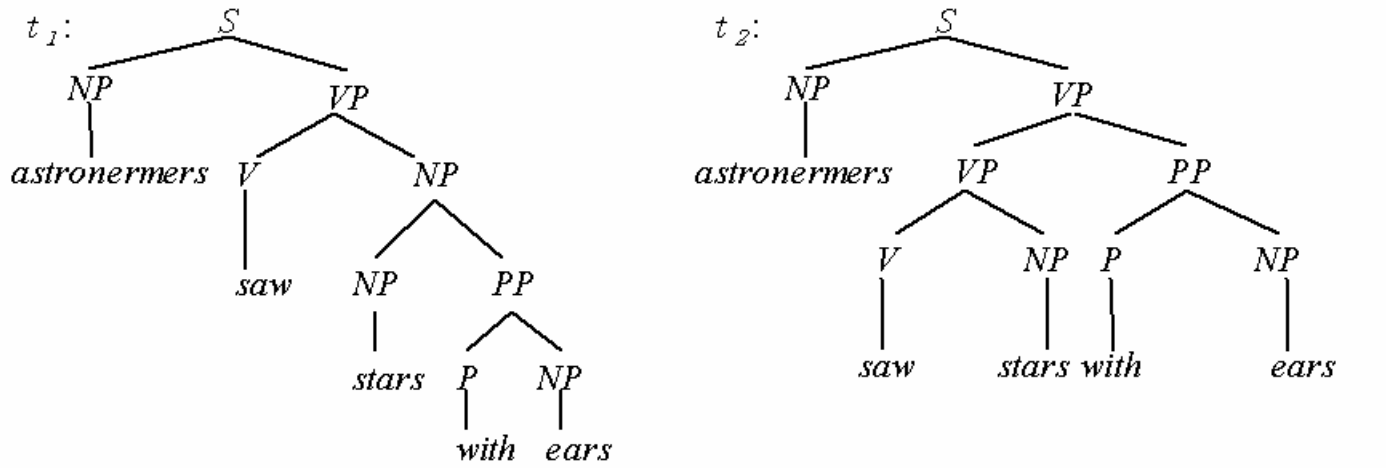


人工语言的句法分析

- 为了某种交流目标而经由人工定义的语言。例如：各种程序设计语言（C语言）。
- 人工语言的特点是无二义性（或没有歧义）。
- 对于满足某种条件的人工语言，存在快速有效的语法分析方法。
  - LL分析法
  - LR分析法
- 甚至存在语法分析器的自动生成工具，例如Yacc。
  - 给定语言的文法，自动产生相应的语法分析程序
- 语法分析 or 句法分析

歧义

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



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

自然语言的句法分析

- 通常句子越长，歧义现象越严重。以NP 为例

  NP PP PP (2)

  NP PP PP PP (5)

  NP PP PP PP PP(14)

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

- 由于句法歧义，成熟的用于分析人工语言的句法分析算法不能直接用于自然语言的句法分析。
- 对于存在歧义的句子，通常在具体的上下文环境中，只有一种分析结果是正确的，句法排歧指的是根据各种知识，选择正确分析结果的过程。
- 基于上下文无关文法的句法分析器应能产生一个句子所有可能的句法分析树。

句法分析算法

- 自顶向下的句法分析
  - 始于开始符号 *S*
  - 利用重写规则进行推导
  - 直到推导出待分析的句子
- 自底向上的句法分析
  - 始于待分析的句子
  - 逆向利用规则进行归约
  - 直到归约出开始符号 *S*
- 非确定性 --- 回溯或并行

[1] $s \rightarrow np\ vp$

[2] $np \rightarrow n$

[3] $vp \rightarrow v'$

[4] $vp \rightarrow v'\ np$

[5] $v' \rightarrow v\ \ u$

[6] $n \rightarrow$ 曹操

[7] $v \rightarrow$ 打败

[8] $u \rightarrow$ 了

[9] $n \rightarrow$ 周瑜

自顶向下

- 采用自顶向下分析的方法分析“周瑜打败了曹操”

(1) *s*

(2) *s* ⇒ *np vp* （使用[1]号规则）

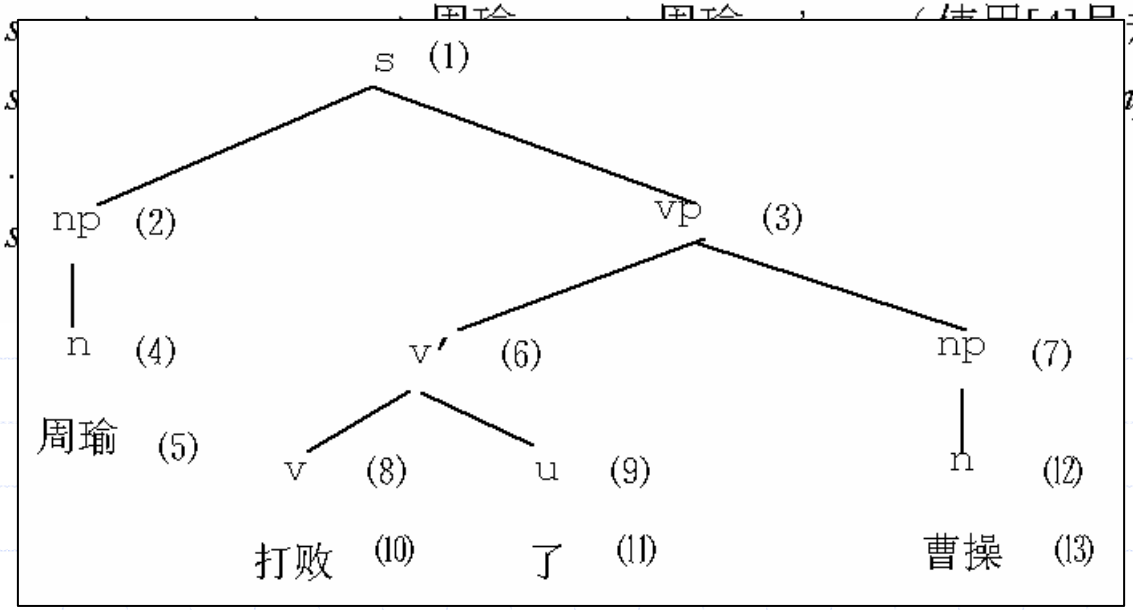
(3) *s* ⇒ *np vp* ⇒ *n vp* （使用[2]号规则）

(4) *s* ⇒ *np vp* ⇒ *n vp* ⇒ 周瑜 *vp* （使用[9]号规则）

(5) *s* ⇒ 周瑜 *v np* （使用[4]号规则）

(6) *s* ⇒ 周瑜 *v n* ⇒ 周瑜 打败 *np* （使用[5]号规则）

(7) *s* ⇒ 周瑜 打败 了 曹操



自底向上

- 采用自底向上分析的方法分析“周瑜打败了曹操”

(1) 周瑜 打败 了 曹操 ⇐ *n* 打败 了 曹操 (使用[9]号规则归约)

(2) *n* 打败 了 曹操 ⇐ *np* 打败 了 曹操 (使用[2]号规则归约)

(3) *np* 打败 了 曹操 ⇐ *np v* 了 曹操 (使用[7]号规则)

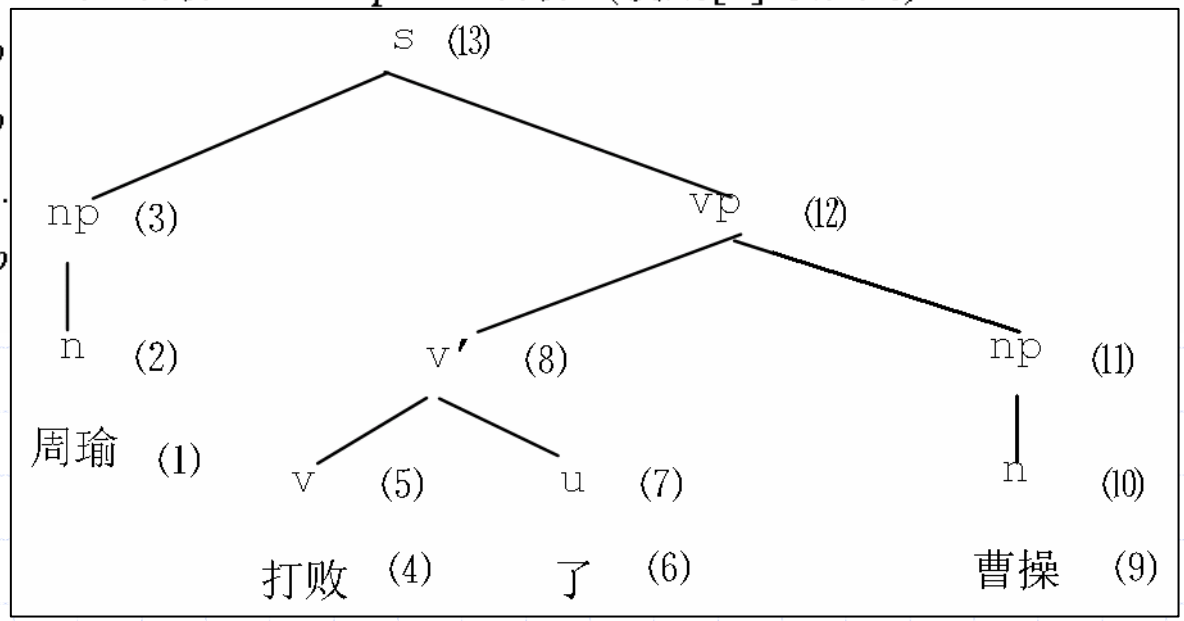
(4) *np v* 了 曹操 ⇐ *np v u* 曹操 (使用[8]号规则)

(5) *np*

(6) *np*

(7) …

np



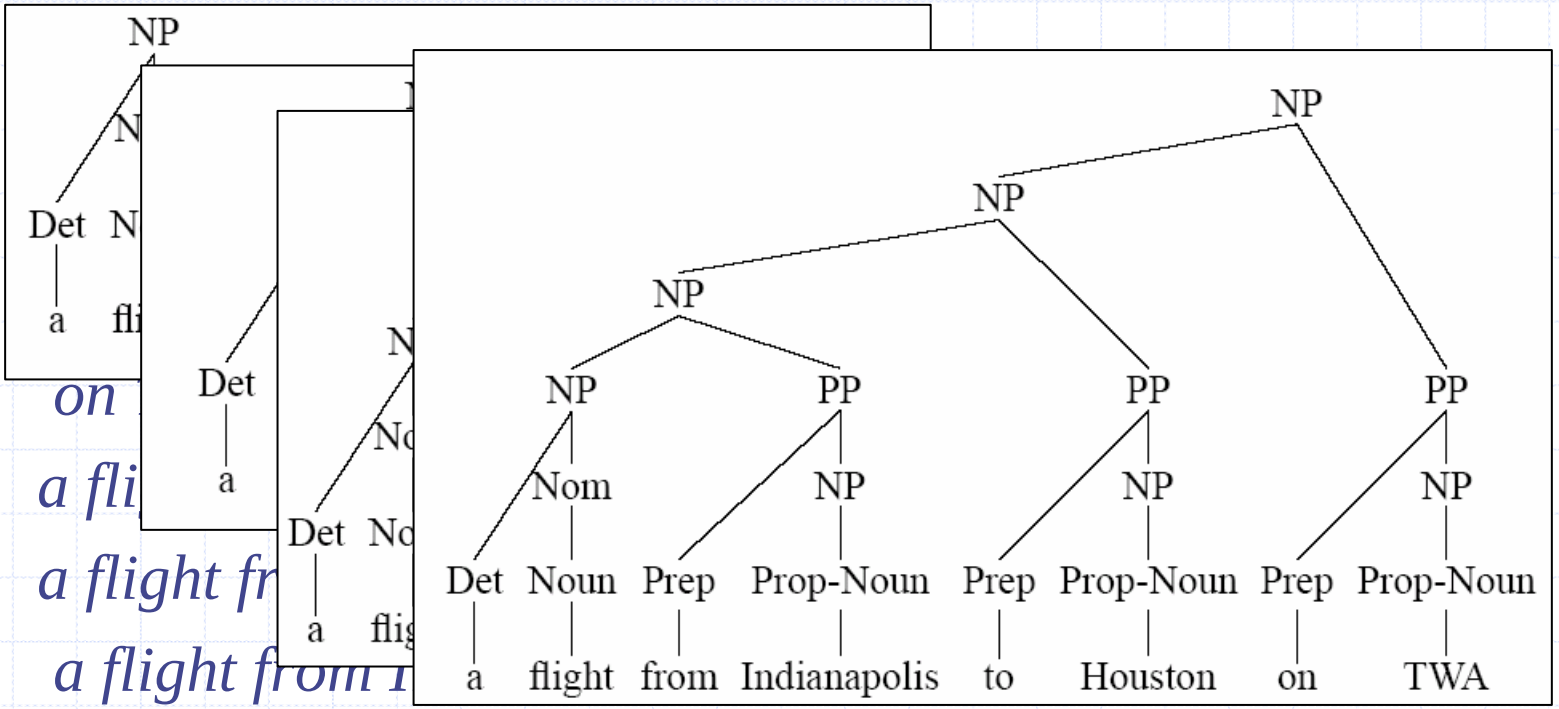
自顶向下与自底而上的结合

- 对于自顶向下的分析方法而言，当有多个重写规则可用时，如何避免选择导致分析失败的规则？
- 对于自底而上的分析方法而言，如果有多种可能的归约，如何避免导致分析失败的归约?
- 可以融合自顶向下、自底而上的方法？
  - 以一种方法为作为主要方法。
  - 另一种方法为辅助。

效率问题

- 回溯导致效率低下，表现为反复分析某些子树。例如，用自顶向下的分析方法分析下面的短语

  a flight from Indianapolis to Houston on TWA



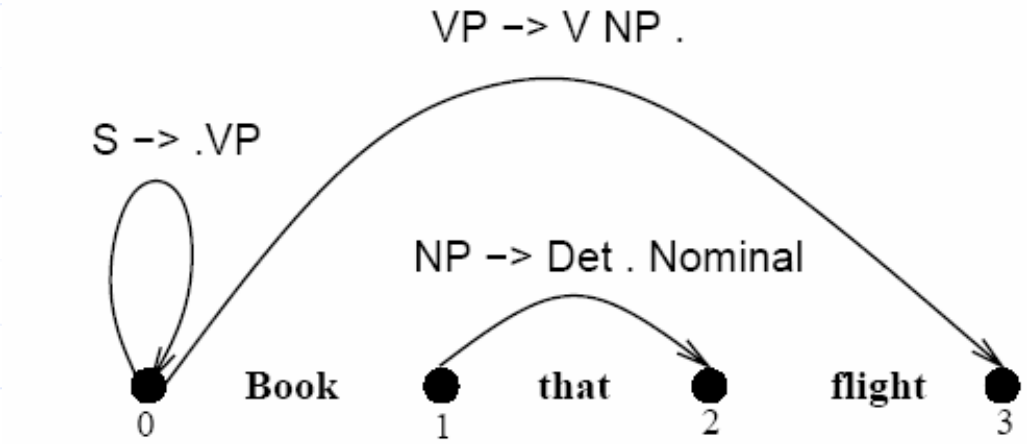
句法分析算法

- 目前已经提出了多种句法分析算法，这些算法都可以相对有效地完成句法分析。
  - Earley分析算法
  - 广义LR分析算法
  - 基于chart的分析算法
  - CYK分析算法
  - ……

Earley算法

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

Earley算法



- 每一个状态对应图中一条边(弧)。如：

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

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

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

- 图中的每个位置对应一个状态表，分别记作

  chart[0], *chart*[1], ……, *chart*[*N*]

Earley算法

- 三种基本操作
  - PREDICTOR()，作用于一个点号后是非终结符号(不包括词类标记)的状态，并产生新的状态。例如：

    对于*chart*[0]中状态*S* →·*VP*,[0.0]，应用PREDICTOR() 在*chart*[0]中产生下列新状态

    VP →·*Verb*,[0,0]

    VP →·*Verb NP*,[0,0]
  - SCANNER()，作用于一个点号后是词类标记的状态，并产生新状态。例如：

    对于*chart*[0]中的状态*VP* →·*Verb NP*,[0,0]，若PARTS-OF-SPEECH(*word*[0])为*Verb*，应用SCANNER()，在chart[1]中产生下面的新状态

    VP →*Verb* · *NP*,[0,1]

Earley算法

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

  对于chart[3]中的状态*NP* →*Det Nominal*·,[1,3],应用COMPLETER()，此时，如果chart[1]中存在状态*VP* →*Verb*·*NP*,[0,1], 则在chart[3]中产生新状态*VP* →*Verb NP*·,[0,3]。

- 初始化，在*chart*[0]加入一个初始状态γ →·*S*, [0,0]。
- 终止条件，若在*chart*[*N*+1]出现状态*S*→α ·,[0,*N*],分析成功。

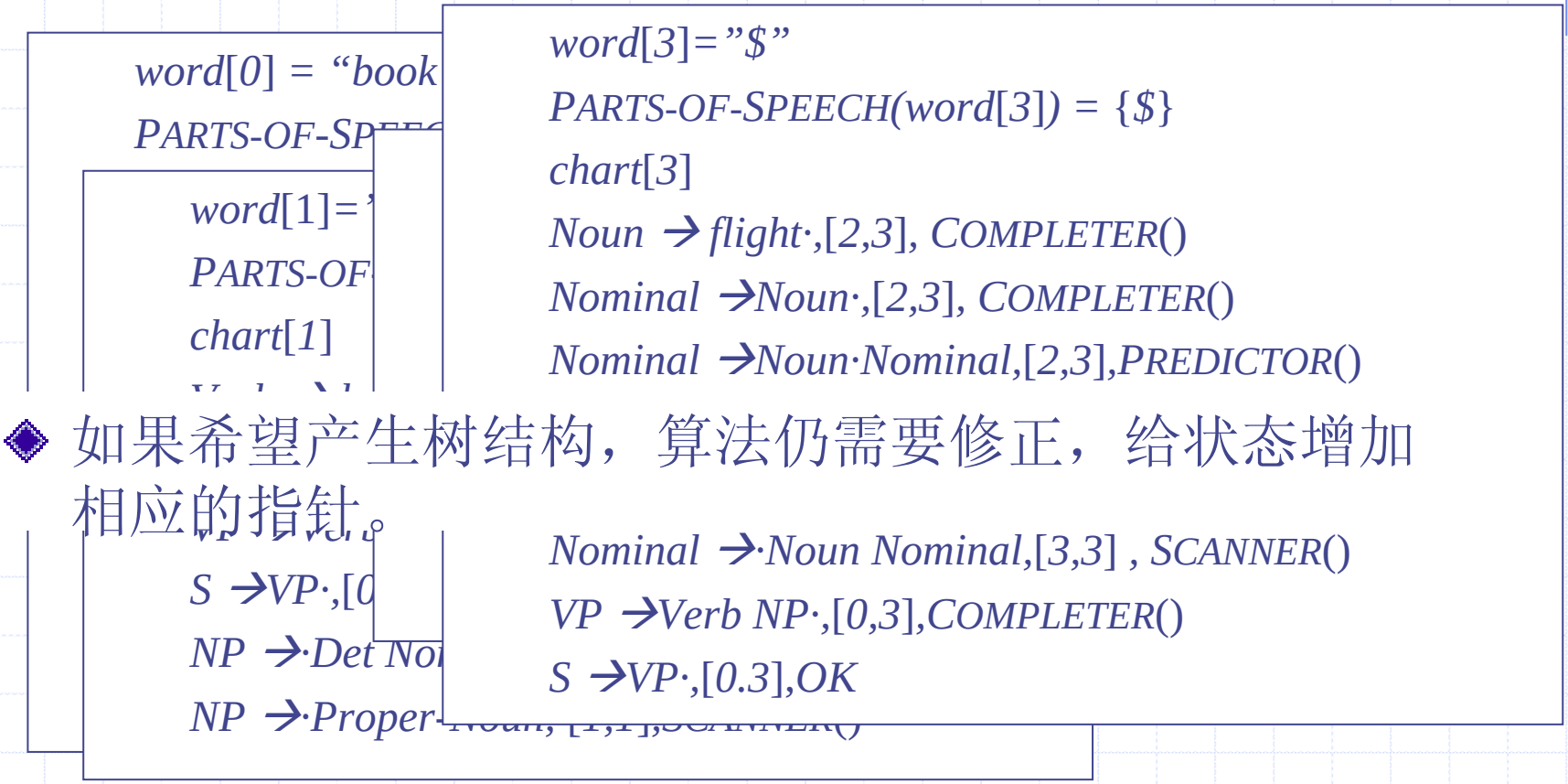
Earley算法

此处缺一个点号

```
function EARLEY-PARSE(words, grammar) returns chart

  ENQUEUE((γ → • S, [0,0]), chart[0])
  for i ← from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
          PREDICTOR(state)
      elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
          SCANNER(state)
      else
          COMPLETER(state)
    end
  end
  return(chart)

procedure PREDICTOR((A → α • B β, [i, j]))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE((B → • γ, [j, j]), chart[j])
    end

procedure SCANNER((A → α • B β, [i, j]))
    if B ⊂ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE((B → word[j], [j, j + 1]), chart[j+1])

procedure COMPLETER((B → γ •, [j,k]))
    for each (A → α • B β, [i, j]) in chart[j] do
        ENQUEUE((A → α B • β, [i,k]), chart[k])
    end

procedure ENQUEUE(state, chart-entry)
    if state is not already in chart-entry then
        PUSH(state, chart-entry)
    end
```

Earley算法

- 分析实例，例如分析句子*book that flight*



- 如果希望产生树结构，算法仍需要修正，给状态增加相应的指针。

标准LR算法

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

标准LR算法

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

标准LR算法

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

标准LR算法

算法：LR 分析算法

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

广义LR算法

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

[0] $S' \rightarrow S$

[1] $S \rightarrow NP\ VP$	[2] $VP \rightarrow V$	[3] $VP \rightarrow V\ NP$
[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] $PP \rightarrow Prep\ NP$

广义LR算法

状	ACTION						GOTO			
态	*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				
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		70		

广义LR算法

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

广义LR算法

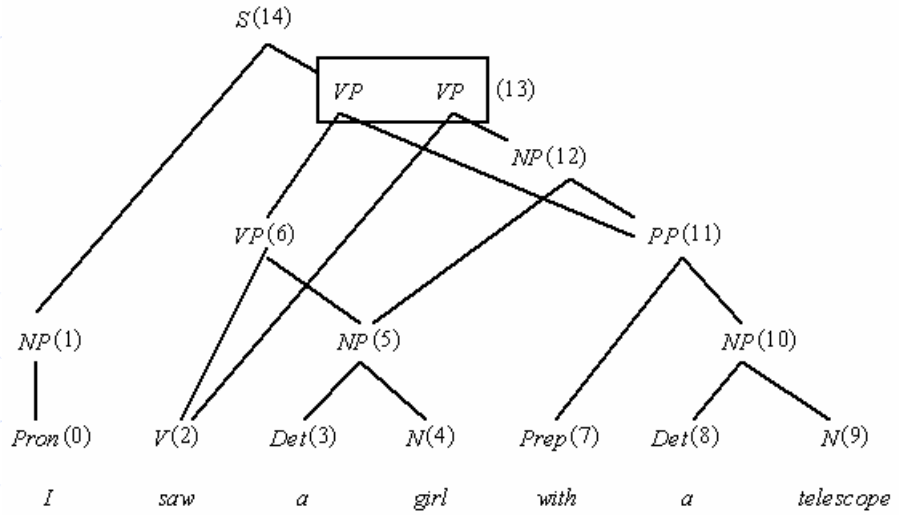
- 子树共享
  - 如果两棵或两棵以上的树具有共同的子树，那么这棵子树就只应该表示一次。
  - 为了构造共享子树，分析过程不再把语法符号入栈，而是将指向共享子树的指针入栈。
  - 当分析器移进一个词时，就用该词和相应的终结符创立叶子结点，如果恰好同一结点已经存在，那么就将已存在结点的指针入栈，而不是另外创立一个结点。当分析器归约时，从栈中弹出指针，创建一个新结点。

广义LR算法

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

广义LR算法

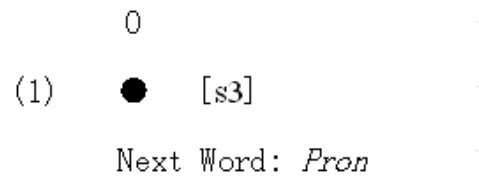
- 压缩共享森林：采用了子树共享技术和局部歧义压缩技术后，得到的分析结果被称为压缩共享森林。



广义LR算法

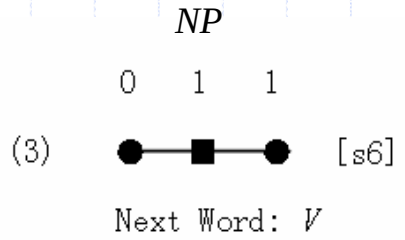
- 广义LR分析算法除上述改进之外，其分析过程同LR分析算法基本相同。
- 分析实例： *I saw a girl with a telescope*
  Pron V Det N Prep Det N $

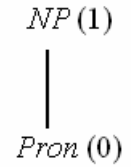
下图中，●表示图结构栈中的状态，■表示压入栈中的指针

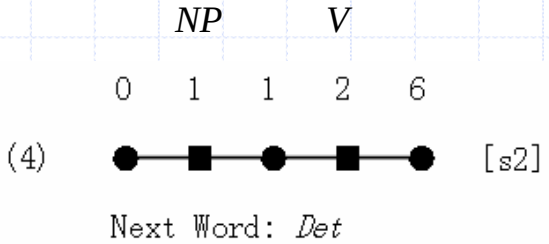


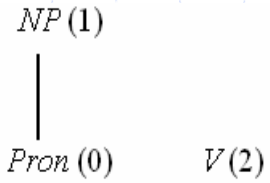
[7] *NP* → *Pron*

广义LR算法

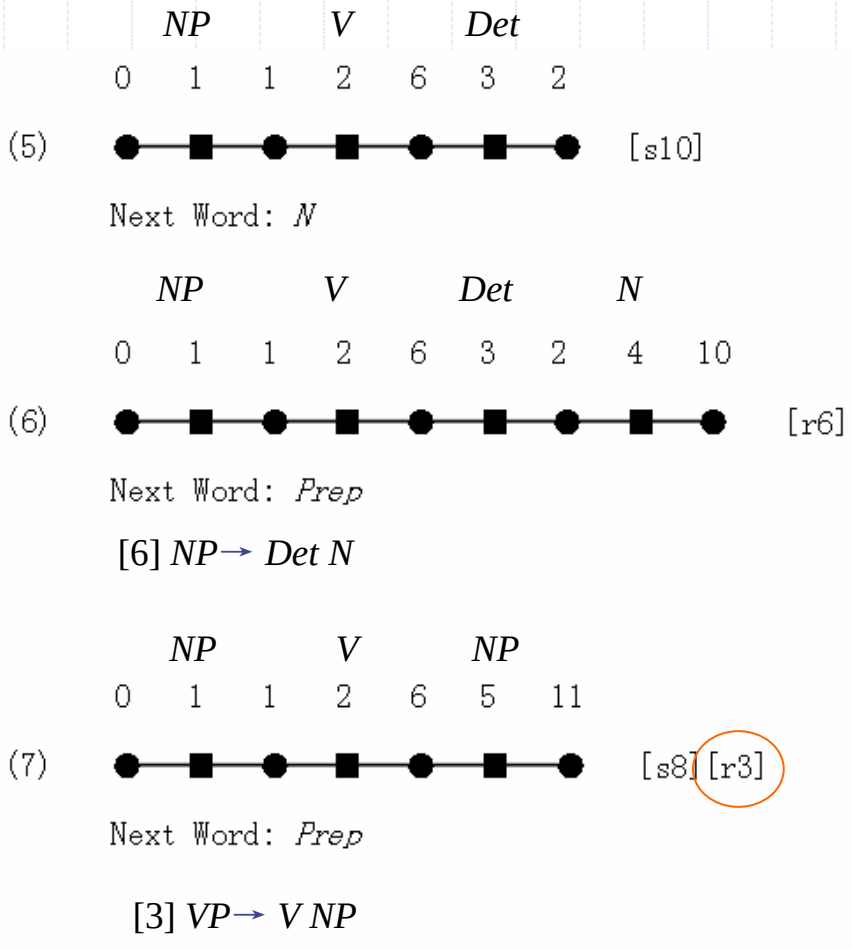

NP
0 1 1
(3)
[s6]
Next Word: V

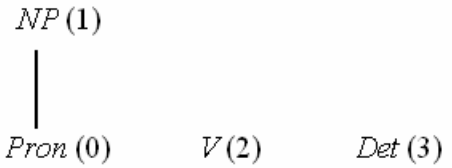

NP (1)
Pron (0)

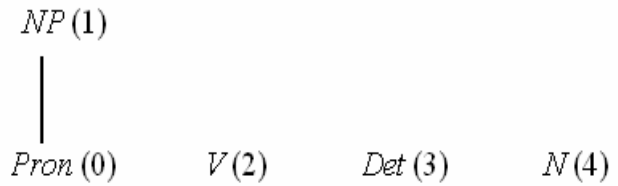

NP V
0 1 1 2 6
(4)
[s2]
Next Word: Det

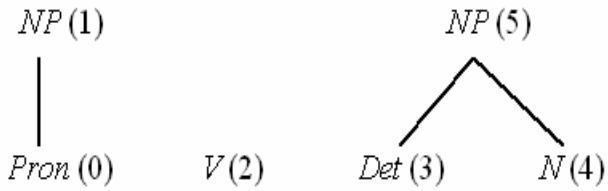

NP (1)
Pron (0)
V (2)

广义LR算法

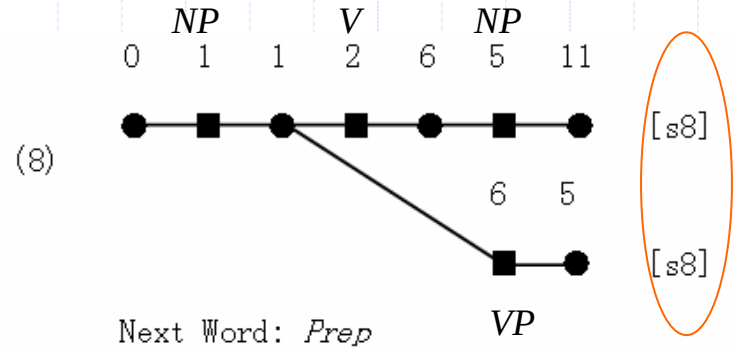

NP V Det
0 1 1 2 6 3 2
(5) [s10]
Next Word: N
NP V Det N
0 1 1 2 6 3 2 4 10
(6) [r6]
Next Word: Prep
[6] NP→ Det N
NP V NP
0 1 1 2 6 5 11
(7) [s8] [r3]
Next Word: Prep
[3] VP→ V NP

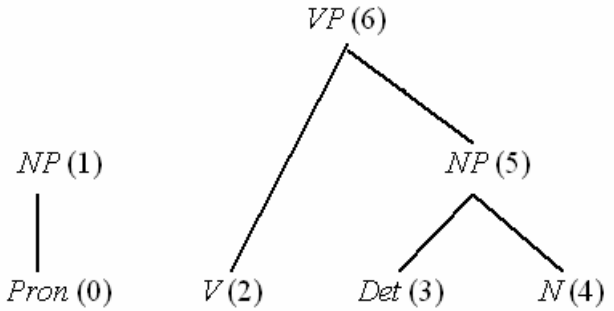

NP (1)
Pron (0)
V (2)
Det (3)

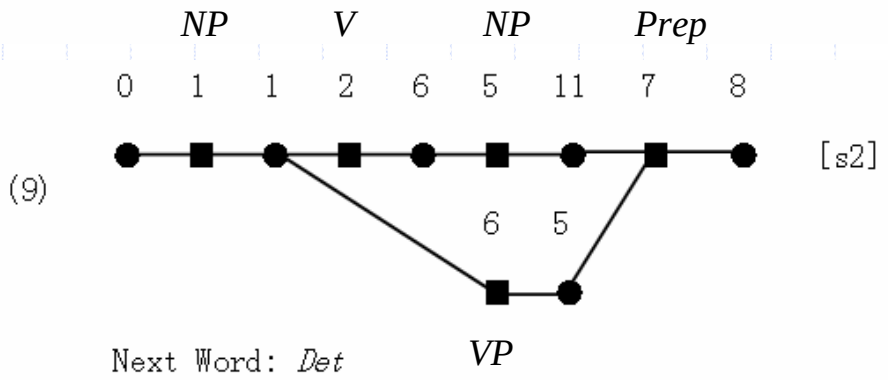

NP (1)
Pron (0)
V (2)
Det (3)
N (4)

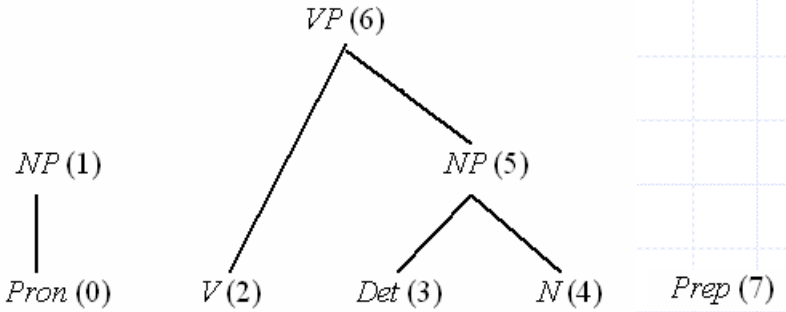

NP (1)
Pron (0)
V (2)
NP (5)
Det (3)
N (4)

广义LR算法

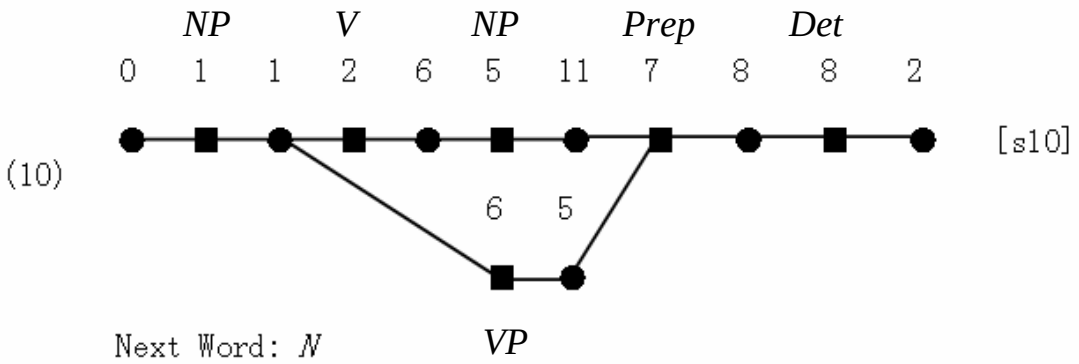

NP V NP
0 1 1 2 6 5 11
(8)
[s8]
6 5
[s8]
Next Word: Prep
VP

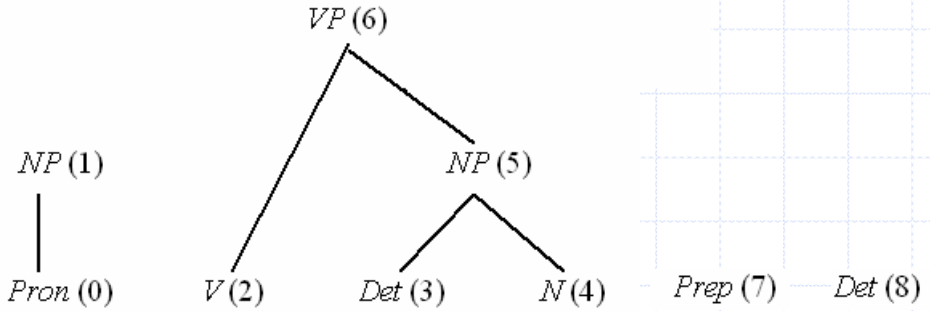

NP (1)
Pron (0)
VP (6)
NP (5)
V (2)
Det (3)
N (4)


NP V NP Prep
0 1 1 2 6 5 11 7 8
(9)
[s2]
6 5
Next Word: Det
VP

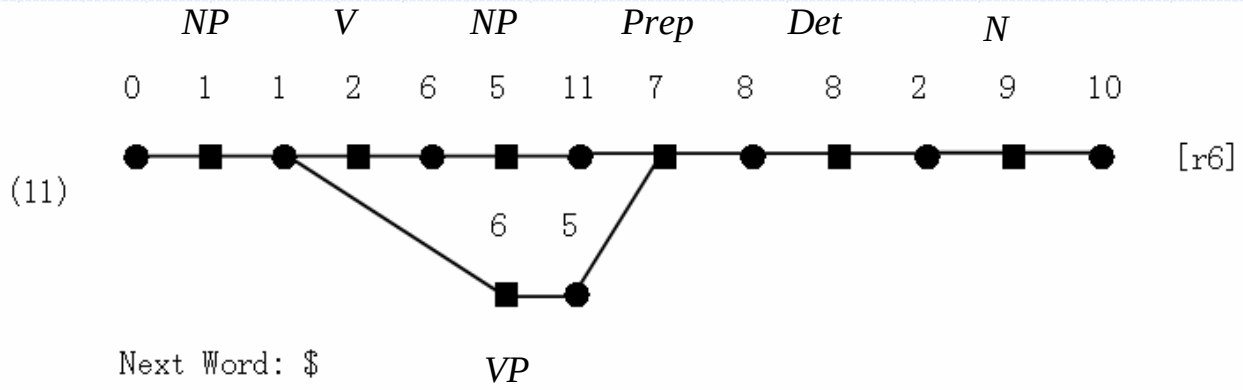

NP (1)
Pron (0)
VP (6)
NP (5)
V (2)
Det (3)
N (4)
Prep (7)

广义LR算法

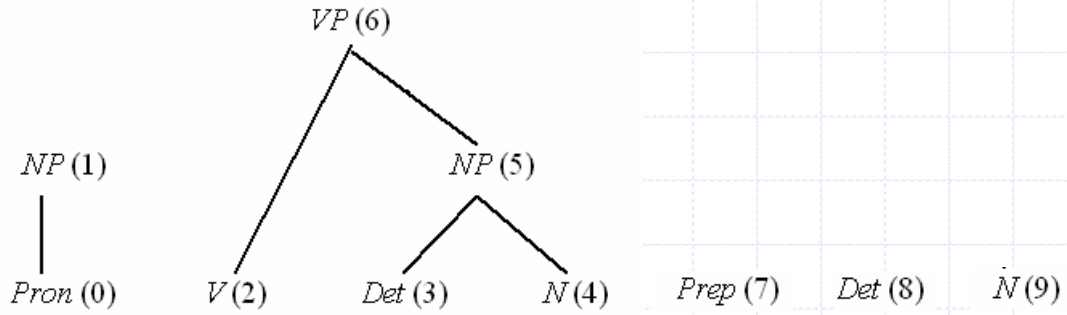

NP V NP Prep Det
0 1 1 2 6 5 11 7 8 8 2
(10)
[s10]
6 5
Next Word: N
VP


VP (6)
NP (1)
NP (5)
Pron (0)
V (2)
Det (3)
N (4)
Prep (7)
Det (8)

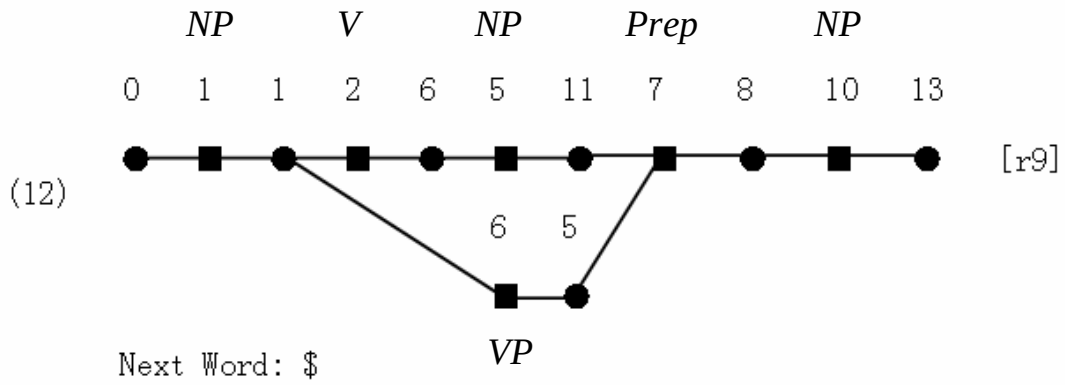
广义LR算法



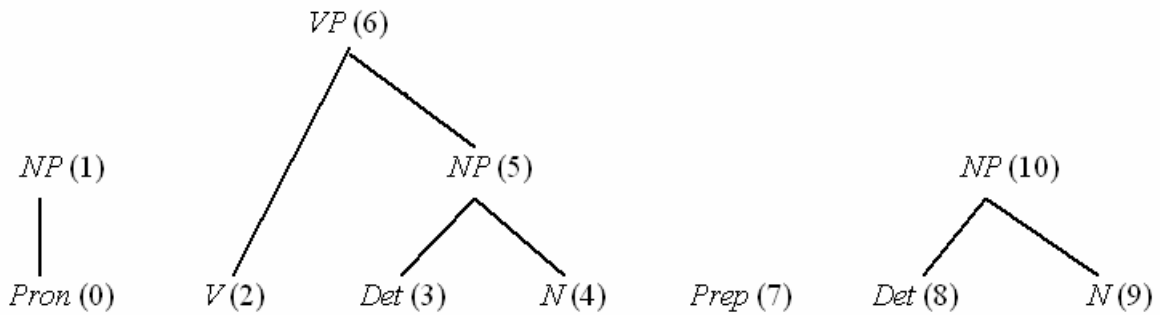
[6] *NP*→ *Det N*



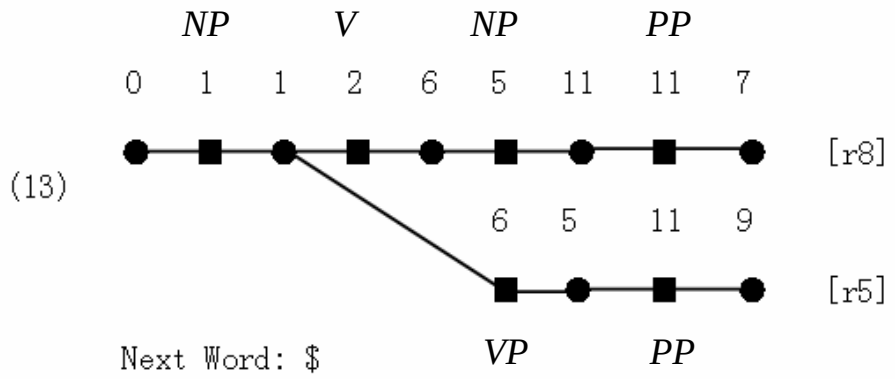
广义LR算法



[9] *PP*→ *Prep NP*

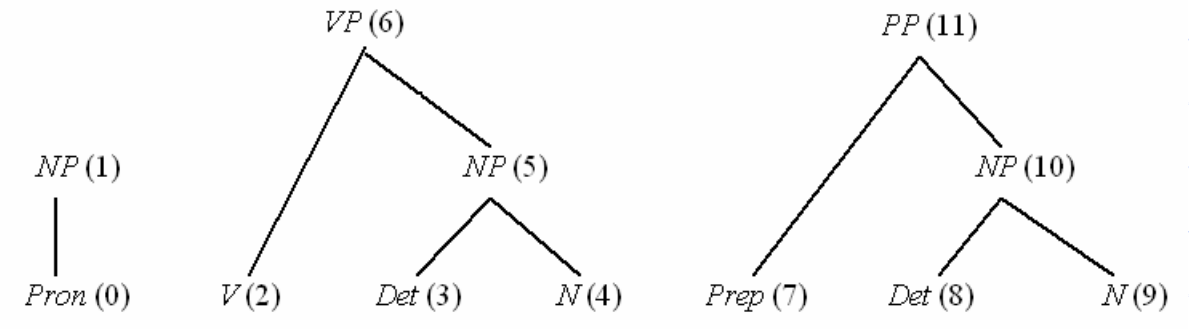


广义LR算法

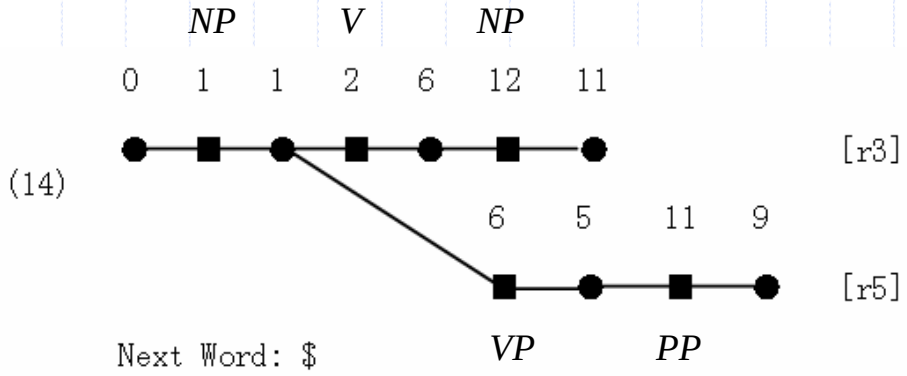


[8] *NP*→ *NP PP*

[5] *VP*→ *VP PP*

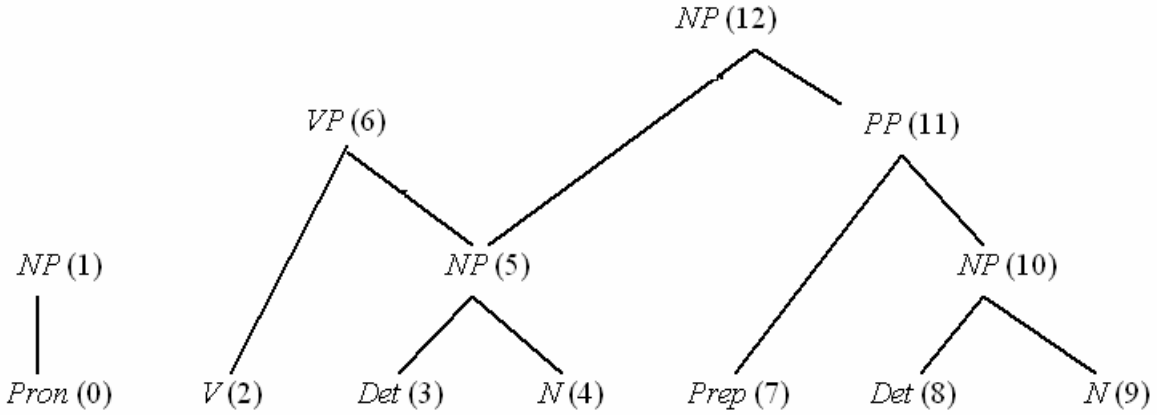


广义LR算法

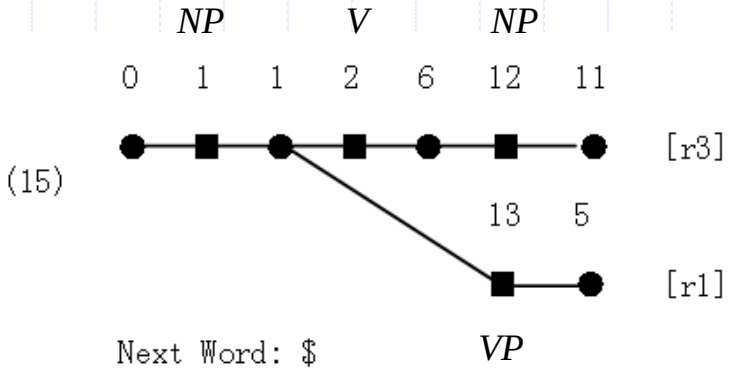


[3] *VP*→ *V NP*

[5] *VP*→ *VP PP*

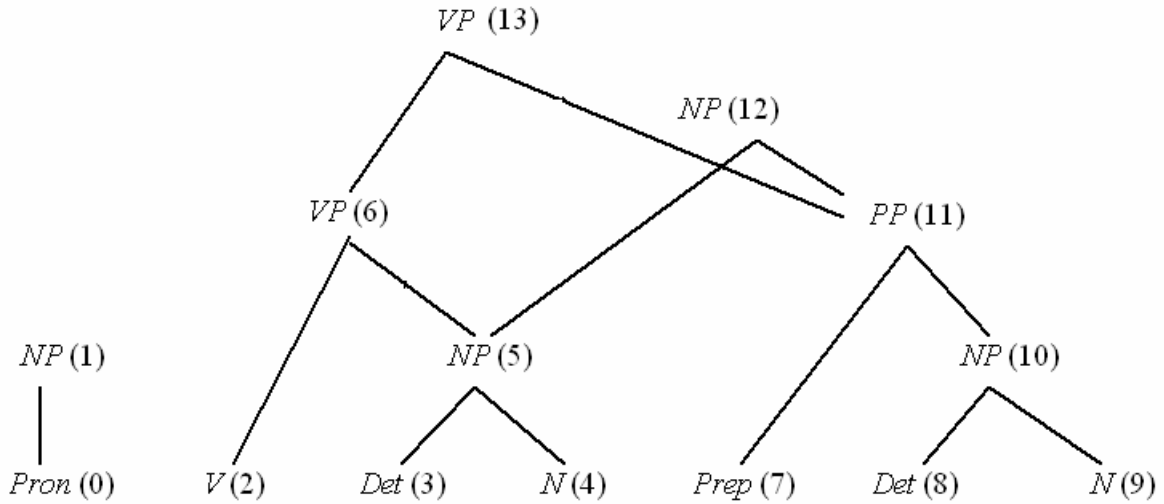


广义LR算法

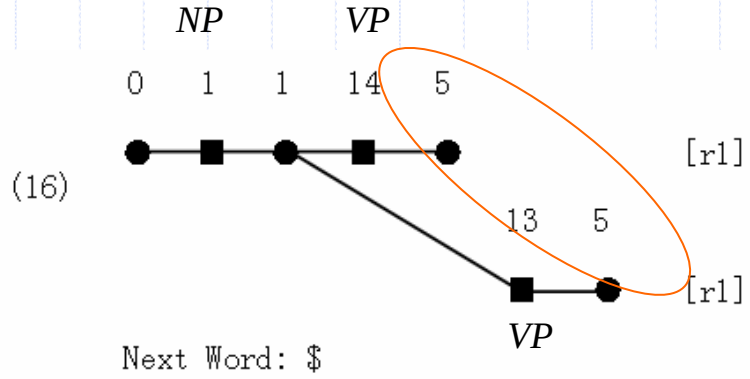


[3] *VP*→ *V NP*

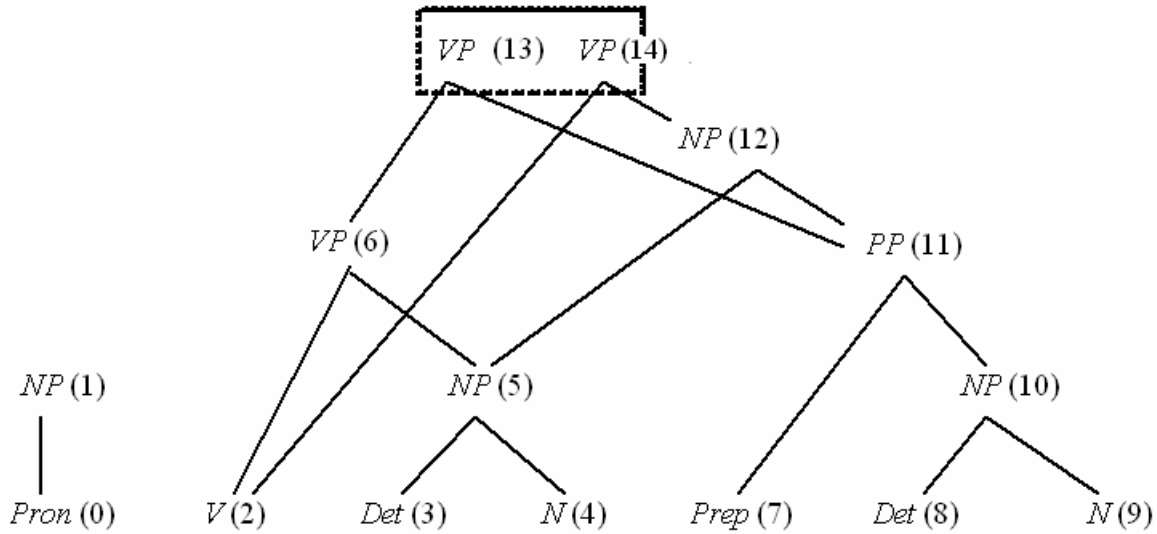
[1] *S*→ *NP VP*



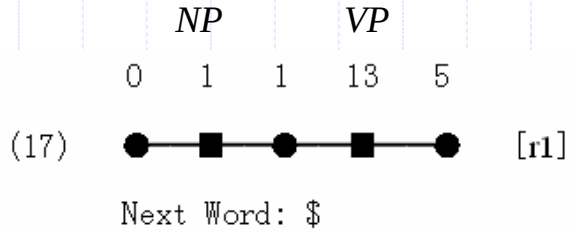
广义LR算法



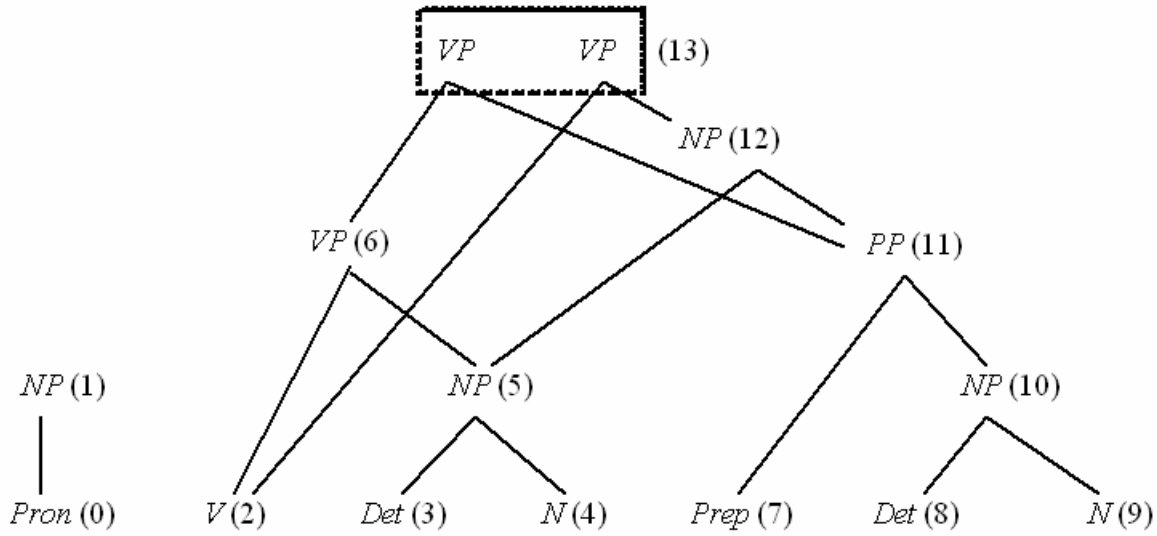
[1] *S*→ *NP VP*



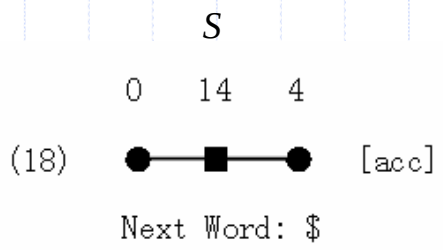
广义LR算法

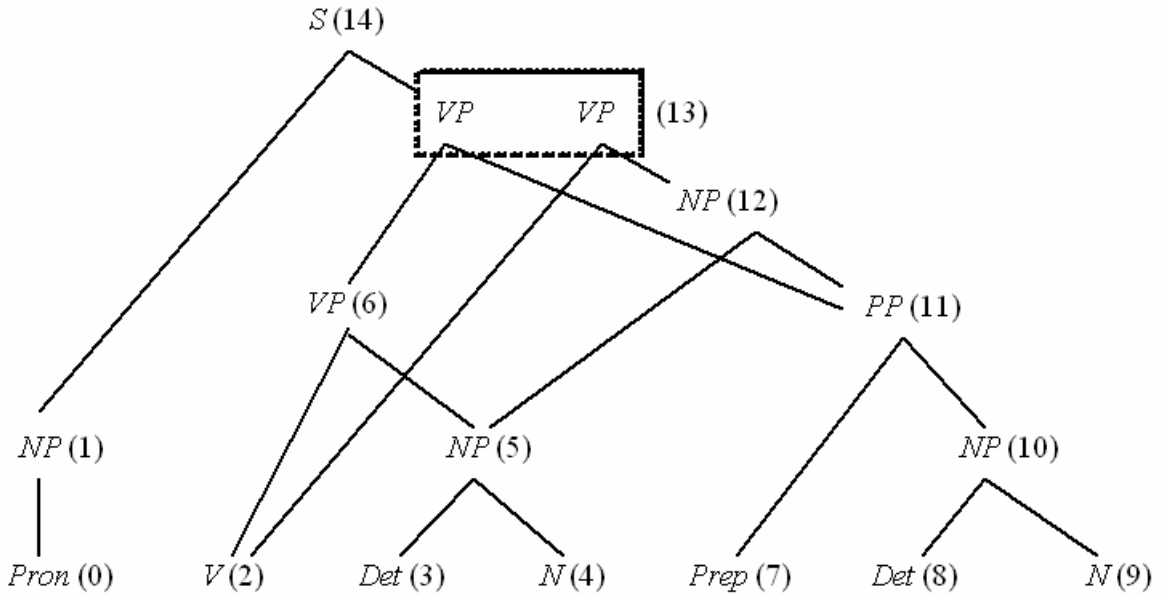


[1] *S*→ *NP VP*



广义LR算法


S
0 14 4
(18)
[acc]
Next Word: $


S (14)
VP VP (13)
NP (12)
VP (6)
PP (11)
NP (1)
NP (5)
NP (10)
Pron (0)
V (2)
Det (3)
N (4)
Prep (7)
Det (8)
N (9)