



句法分析

杨沐昀

教育部-微软语言语音重点实验室

MOE-MS Joint Key Lab of NLP and Speech (HIT)

句法分析概述

- 定义：判断单词串是否属于某个语言(recognition)，如果是，则给出其（树）结构(parsing)

子问题1：语言体系的形式化描述

子问题2：语言结构的分析算法

句法分析概述

- 一般而言，句法结构分析的任务有三个：
 1. 判断输入的字符串是否属于某种语言
 2. 消除输入句子中词法和结构等方面的歧义
 3. 分析输入句子的内部结构，如成分构成、上下文关系等。
如果一个句子有多种结构表示，句法分析器应该分析出该句子最有可能的结构
- 实际应用中，通常系统都已经知道或者默认了被分析的句子属于哪一种语言，因此，一般不考虑任务1，而着重考虑2，3的处理问题

句法分析概述

- 给定英语单词：the,can,hold,water的词性信息、及句法规则：

the : art(冠词)

can : n, aux, v(n, v表示名词和动词,aux表示助动词)

hold : v

water : n, v

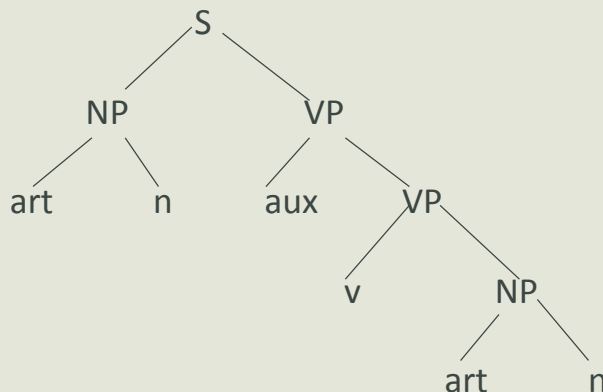
s -> NP VP

NP -> art n

VP -> aux VP

VP -> v NP

- 那么句子The can can hold the water 的分析树如下所示：



语言的描述

- 一般的，描述一种语言可以有三种途径：
 1. 穷举法：把语言中的所有句子都枚举出来。显然，这种方法只适合句子数目有限的语言
 2. 语法/文法描述：语言中的每个句子用严格定义的规则来构造，利用规则生成语言中合法的句子
 3. 自动机法：通过对输入句子进行合法性检验，区别哪些是语言中的句子，哪些不是语言中的句子
- 语法/文法用来精确的描述语言和其结构，自动机则是用来机械地刻画对输入字符串的识别过程

形式语法的定义

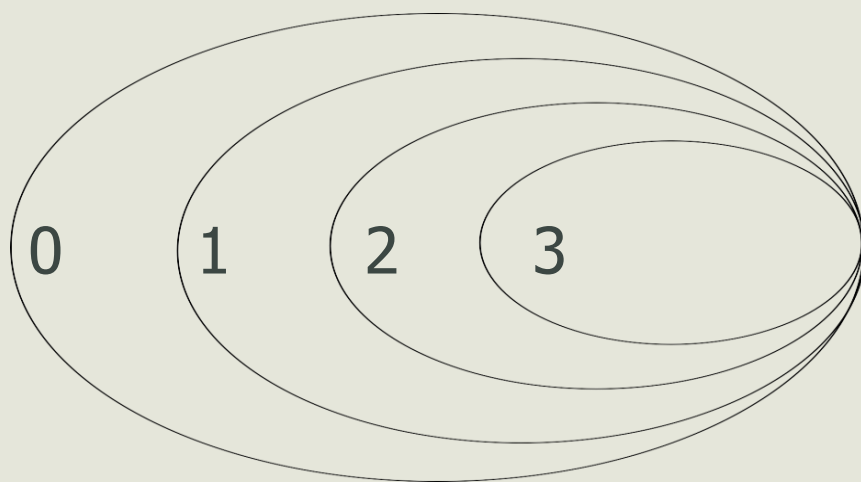
- 形式语法：四元组 $G = \{N, \Sigma, P, S\}$
 - N 是非终结符(non-terminal symbol)的有限集合(有时也称变量集或句法种类集)
 - Σ 是终结符号(terminal symbol)的有限集合, $N \cap \Sigma = \emptyset$
 - (Σ 也经常用T标示)
 - $V = N \cup \Sigma$ 称为总词汇表
 - P 是一组重写规则的有限集合: $P = \{\alpha \rightarrow \beta\}$, 其中 α, β 是由 V 中元素构成的串, 但是 α 中至少应含一个非终结符
 - $S \in N$ 称为句子符或初始符

形式语法：种类

- 正则文法：如果文法G的规则集P中所有规则均满足如下形式： $A \rightarrow Bx$ 或 $A \rightarrow x$,其中 $A, B \in N, x \in \Sigma$,则称该文法G为正则文法，或称3型文法
- 上下文无关文法：如果文法G的规则集P中所有规则均满足如下形式： $A \rightarrow \alpha$,其中 $A \in N, \alpha \in (N \cup \Sigma)^*$ ，则称该文法为上下文无关文法(context-free grammar, CFG)，或称2型文法
- 上下文相关文法：如果文法G的规则集P中所有规则均满足如下形式： $\alpha A \beta \rightarrow \alpha \gamma \beta$ ，其中 $A \in N, \alpha, \beta, \gamma \in (N \cup \Sigma)^*$ ，且 γ 至少包含一个字符，则称G为上下文相关文法(context-sensitive grammar, CSG)，或称1型文法
- 无约束文法：如果文法G的规则集P中所有规则均满足如下形式： $\alpha \rightarrow \beta$ ，其中 $\alpha \in (N \cup \Sigma)^+$ ， $\beta \in (N \cup \Sigma)^*$ ，则称G无约束文法，也称0型文法

形式文法的乔姆斯基层级体系

分级	名称	产生式规则的形式限制
0	PSG	$\alpha \rightarrow \beta$ with $\alpha \in (V_T \cup V_N)^+$ and $\beta \in (V_T \cup V_N)^*$
1	CSG	$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ with $A \in V_N$ and $\alpha_1, \alpha_2 \in (V_T \cup V_N)^*$ and $\beta \in (V_T \cup V_N)^+$
2	CFG	$A \rightarrow \beta$ with $A \in V_N$ and $\beta \in (V_T \cup V_N)^*$
3	RG	$A \rightarrow \beta B$ or $A \rightarrow \beta$ with $A, B \in V_N$ and $\beta \in V_T^*$



G_0 : 无限制重写文法 PSG

G_1 : 上下文相关文法 CSG

G_2 : 上下文无关文法 **CFG**

G_3 : 正则文法 RG

文法、自动机和语言

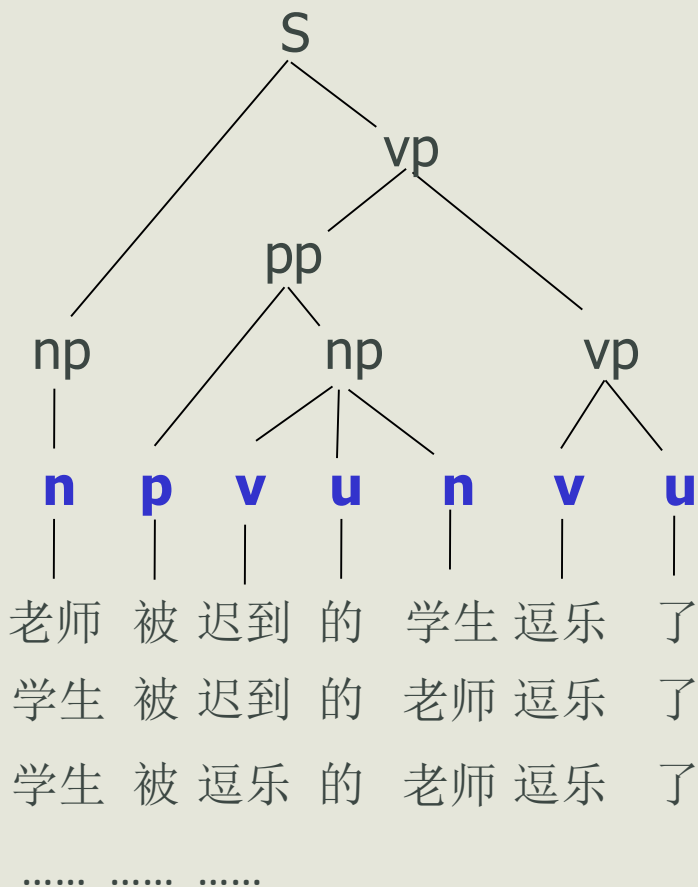
	文法	自动机	语言	复杂度
0型	无约束文法	图灵机	递归可枚举语言	半可判定
1型	上下文相关文法	线性界限自动机	上下文相关语言	NP完全
2型	上下文无关文法	下推自动机	上下文无关语言	多项式
3型	正则文法	有限自动机	正则语言	线性

文法与自然语言

- 上下文无关文法使用最普遍
 - 正则语法描述能力太弱、上下文相关语法计算复杂度太高
- 计算复杂度：上下文无关文法的复杂度是多项式的，其复杂度可以忍受
- 描述能力：上下文无关文法不足以描述自然语言，自然语言中上下文相关的情况非常常见；
- 为弥补上下文无关文法描述能力的不足，需要加上一些其他手段扩充其描述能力
- 思考题：这种现实主义的选择是否意味着放弃了科学？会不会造成计算语言学与NLP分道扬镳？

例：用CFG描述自然语言

1. $S \rightarrow np \ vp$
2. $np \rightarrow vp \ u \ np$
3. $vp \rightarrow pp \ vp$
4. $vp \rightarrow vp \ u$
5. $pp \rightarrow p \ np$
6. $np \rightarrow n$
7. $vp \rightarrow v$
8. $n \rightarrow \text{老师} \mid \text{学生} \dots$
9. $v \rightarrow \text{迟到} \mid \text{逗乐} \dots$
10. $p \rightarrow \text{被} \dots$
11. $u \rightarrow \text{的} \mid \text{了} \dots$



句法分析算法的主要处理策略

- 自顶向下、自底向上是两种控制策略
- 深度优先和广度优先是搜索策略，都适用两种控制策略
 - 探索多种可能性时选择的排队策略(栈或队列)
- 自左至右和自右至左(甚至“中间开花”)是常用的两种扫描策略
- 当文法没有二义性时,甚至假设句子理解没有二义性时,可以采用确定性算法(无回溯);对自然语言一般不行

自底向上和自顶向下

自底向上（bottom-up） 基于规约的方法

从待分析字符串开始，用待分析字符串去匹配CFG规则箭头的右部字符，匹配成功后替换为左部字符，直到S

自顶向下（top-down） 基于预测的方法

从CFG规则中的S规则开始，将CFG规则箭头左部的符号展开，直到形成以终结符开始的序列，用该序列去匹配待分析字符串，直到完全匹配上

自顶向下和自底向上

Top - Down

s

s -> np, vp

s -> pron, vp

s -> John, vp

s -> John, v, np

s -> John, hit, np

s -> John, hit, det, n

s -> John, hit, the, n

s -> John, hit, the, cat

Bottom - Up

John, hit, the, cat

pron, hit, the, cat

pron, v, the, cat

pron, v, det, cat

pron, v, det, n

np, v, det, n

np, v, np

np, vp

s

移进 - 归约算法

- 基本数据结构：**堆栈**

- 四种操作

- **移进(Shift)**：从句子左端将一个终结符移到栈顶
- **归约(Reduce)**：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-规约方法分析

- 两种冲突
 - 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
 - 回溯策略：对于互相冲突的各项操作，给出一个选择顺序
 - 移进 - 归约冲突：优先进行归约，如果失败再尝试移进
 - 归约 - 归约冲突：规则事先排序，优先执行排在前面的规则
 - 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

实例:移进-规约分析过程

- 句法规则

- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

- 输入句子

我是上级派来的
N V N V V de

实例:移进-规约分析过程

栈	输入	操作
#	N V N V V de	sh

文法规则：

- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

实例:移进-规约分析过程

栈	输入	操作
#	N V N V V de	sh
# N	V N V V de	r(2) [or sh?]
# NP	V N V V de	sh
# NP V	N V V de	sh
# NP V N	V V de	r(2) [or sh?]
# NP V NP	V V de	r(5) [or sh?]
# NP VP	V V de	r(1) [or sh?]
# S	V V de	sh
# S V	V de	sh
# S V V	de	sh [not r(6)]

文法规则：

- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

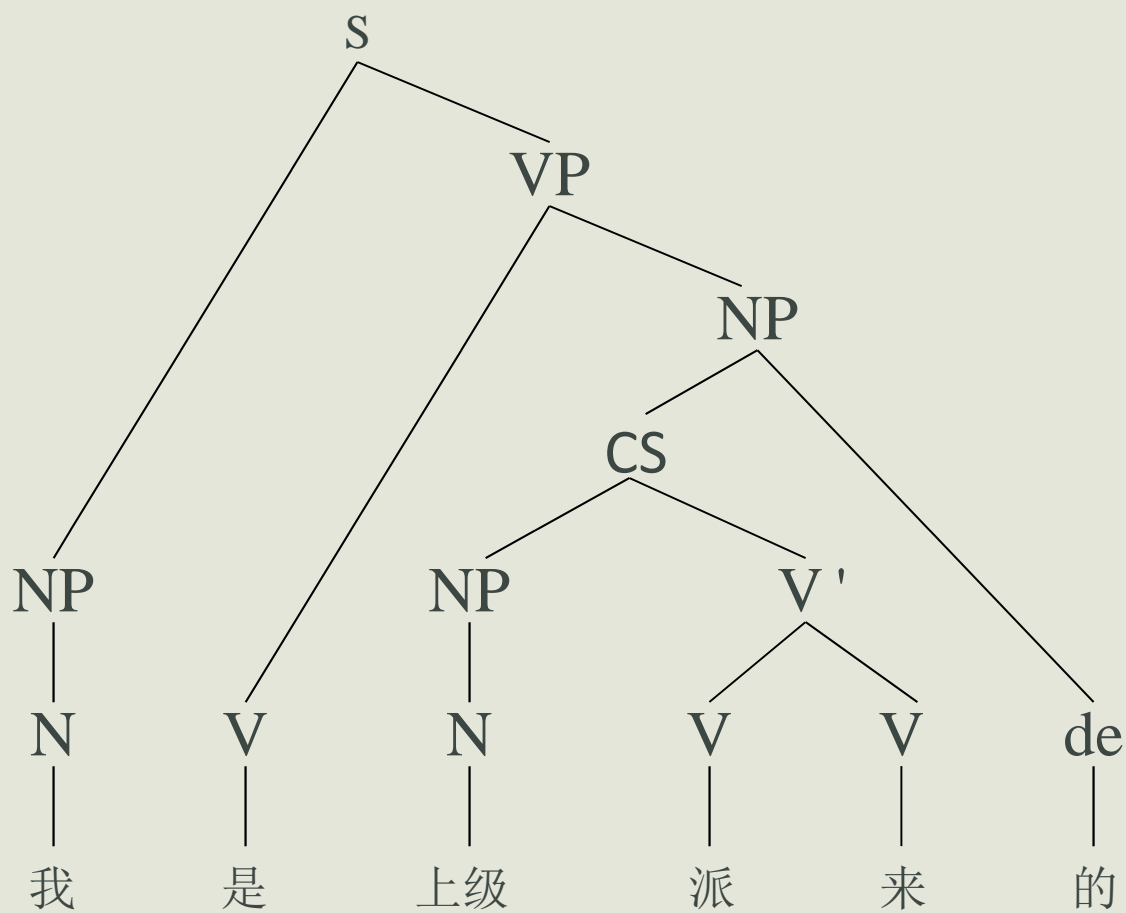
实例:移进-规约分析过程

栈	输入	操作
#	N V N V V de	sh
# N	V N V V de	r(2) [or sh?]
# NP	V N V V de	sh
# NP V	N V V de	sh
# NP V N	V V de	r(2) [or sh?]
# NP V NP V	V de	sh [not r(5)]

文法规则：

- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

产生的句法结构树



移进-规约方法分析

- 移进 - 归约算法是一种**自底向上**的分析算法
- 为了得到正确的分析结果，可以在每次分析失败时都强制性回溯，直到分析成功
- 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

线图分析算法 (Chart parsing)

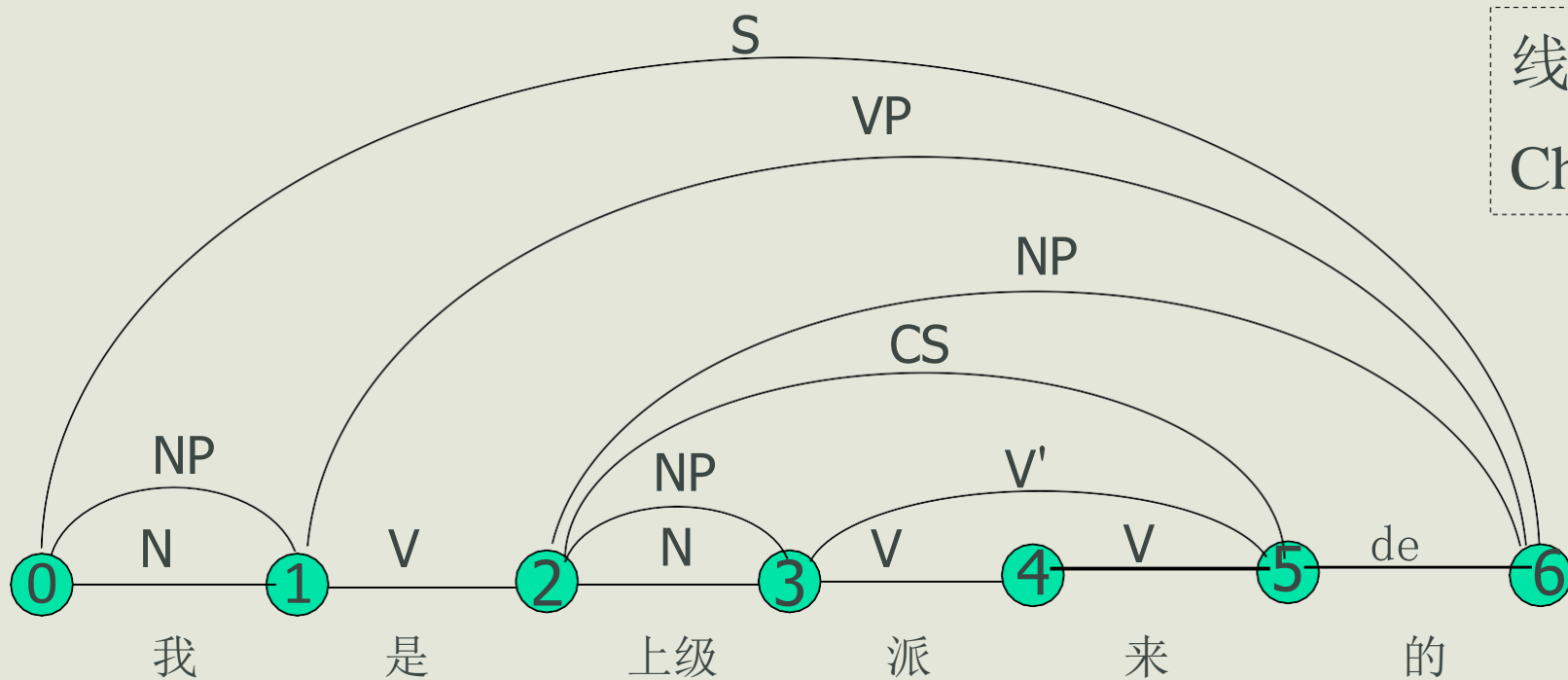
■ General Principles:

A Bottom-Up parsing method

- Construct a parse starting from the input symbols
- Build constituents from sub-constituents
- When all constituents on the RHS of a rule are matched, create a constituent for the LHS of the rule

The *Chart* is a data structure that allows storing of partial analyses, so that they can be shared. (like dynamic programming)

线图分析图示



基本概念：线图/chart

线图是一组节点(node)和边(edge)的集合

节点：对应着输入字符串中的字符间隔

边：<起点, 终点, 标记>

其中标记为非终结符或终结符

问题：

如何从输入串开始，一步步形成**chart**，使得存在一条边可以覆盖全部节点，并且边上标记为**S**？

点规则 (Dot rule)

- 当考虑自左至右分析时，引入点规则表明分析的状态。
- 点规则由四部分组成：
 - (1) 上下文无关语法规则
 - (2) 圆点 \cdot (圆点左边的部分是已分析的，右边是待分析的)
 - (3) 整数 i : 起点 (已分析子串的起点)
 - (4) 整数 j : 终点 (已分析子串的终点) $i \leq j$

比如： $S \rightarrow NP \cdot VP <0,4>$

表示在寻找 S 的过程中，在词 0 到 4 之间已发现一个 NP ，期望下面有一个 VP

线图分析算法——数据结构

1) chart（线图）

$\{edge_{[i]}\} i=1,2,\dots$ $edge := \langle P1, P2, Label \rangle$

保存分析过程中已经建立的成分和位置

2) agenda（议程表，待处理表）

存放等待加入到chart中的边（edge）

可以栈（stack）结构，或队列（queue）结构实现

3) active arc（活动弧）

如果点不在产生式的最右部，在分析中称为“活动弧”，存放分析过程中的中间状态

Chart算法的过程描述

- 1) 将待分析字符串w置入输入缓冲区，agenda清为空栈；
- 2) 循环，反复执行下面步骤，直至输入缓冲区和agenda均为空
 - a) 若agenda为空，则从输入缓冲区取一个字符，并把该字符及其起止位置(P1, P2)推入agenda栈；
 - b) 若agenda不为空，则从agenda中弹出栈顶的边，该边的起止位置为(P1, P2)，边上标记为L；
 - c) 检查规则集中的规则，对所有形如 $A \rightarrow L \beta$ 这样的规则，在active arc集合中增加一条起止位置为P1, P2，弧上为 $A \rightarrow L \cdot \beta$ 这样的点规则；
 - d) 把从agenda中弹出的标记为L的边，加入到chart中的P1, P2之间；
 - e) 检查所有active arc，如果存在起止位置为P0, P1，且弧上点规则为 $A \rightarrow \alpha \cdot L \beta$ 的active arc，就增加一条新的active arc，起止位置为P0, P2，弧上点规则为 $A \rightarrow \alpha L \cdot \beta$
 - f) 如果一条active arc（起止位置为P0, P2）上点规则形如 $A \rightarrow \alpha L \cdot$ （点号在规则最右端），就将起止位置为P0, P2，边上标记为A的边压入agenda栈。

扫描
移进

归约

预测

agenda中的这条边将为下面的分析指出方向：移进所有右部以A开头的规则（步骤b, c）

Chart算法分析示例

chart
active

“线图”用于存放已经确定的分析结果



“活动弧”用于存放分析的中间结果
(包括尚未确定的, 和已经确定的分析结果)



输入缓冲区

文法规则：

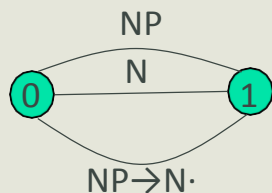
- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

agenda

“议程表”用于
中转当前正在
处理的边

Chart算法分析示例

chart
active



文法规则：

- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

agenda

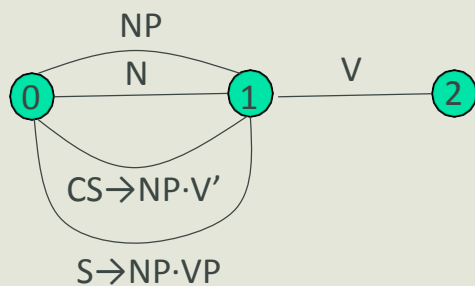


输入缓冲区

(0,1) N

Chart算法分析示例

chart
active
arc



文法规则：

- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

agenda

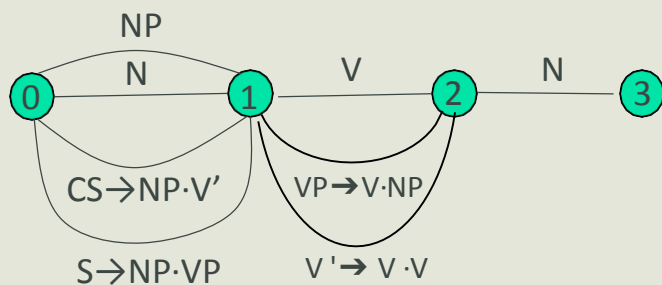
N	V	V	de
---	---	---	----

输入缓冲区

(1,2) V

Chart算法分析示例

chart
active
arc



文法规则：

- (1) $S \rightarrow NP \ VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS \ de$
- (4) $CS \rightarrow NP \ V'$
- (5) $VP \rightarrow V \ NP$
- (6) $V' \rightarrow V \ V$

agenda

v	v	de
---	---	----

输入缓冲区

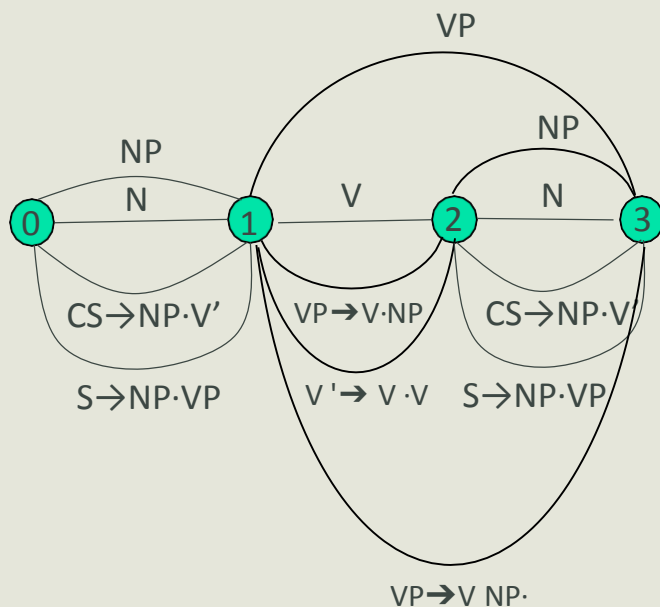
(2,3) N

Chart算法分析示例

c
h
a
r
t

a
c
t
i
v
e

a
r
c



文法规则：

- (1) $S \rightarrow NP \ VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS \ de$
- (4) $CS \rightarrow NP \ V'$
- (5) $VP \rightarrow V \ NP$
- (6) $V' \rightarrow V \ V$

agenda

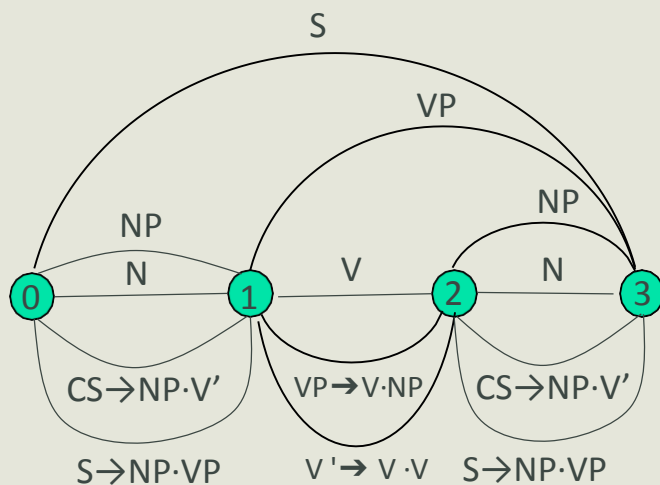
(1,3) VP

v	v	de
---	---	----

输入缓冲区

Chart算法分析示例

chart
active
arc



文法规则：

- (1) $S \rightarrow NP \ VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS \ de$
- (4) $CS \rightarrow NP \ V'$
- (5) $VP \rightarrow V \ NP$
- (6) $V' \rightarrow V \ V$

agenda

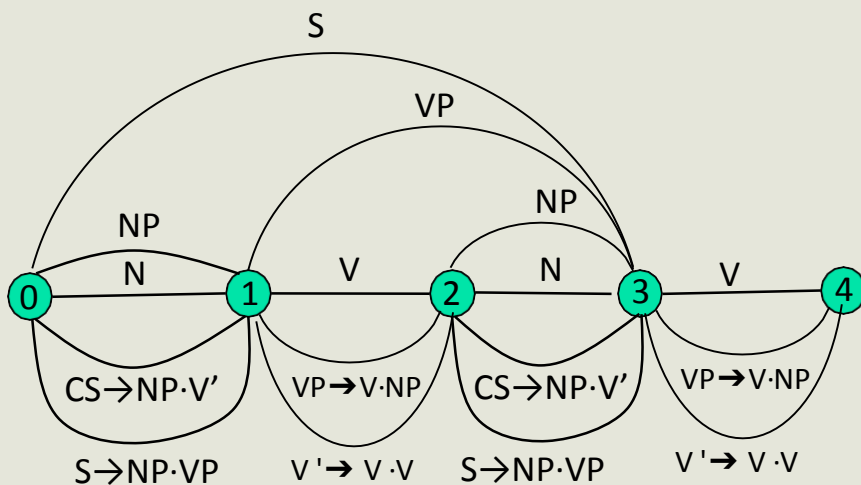
(0,3) S

v	v	de
---	---	----

输入缓冲区

Chart算法分析示例

chart
active
arc



文法规则：

- (1) $S \rightarrow NP \ VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS \ de$
- (4) $CS \rightarrow NP \ V'$
- (5) $VP \rightarrow V \ NP$
- (6) $V' \rightarrow V \ V$

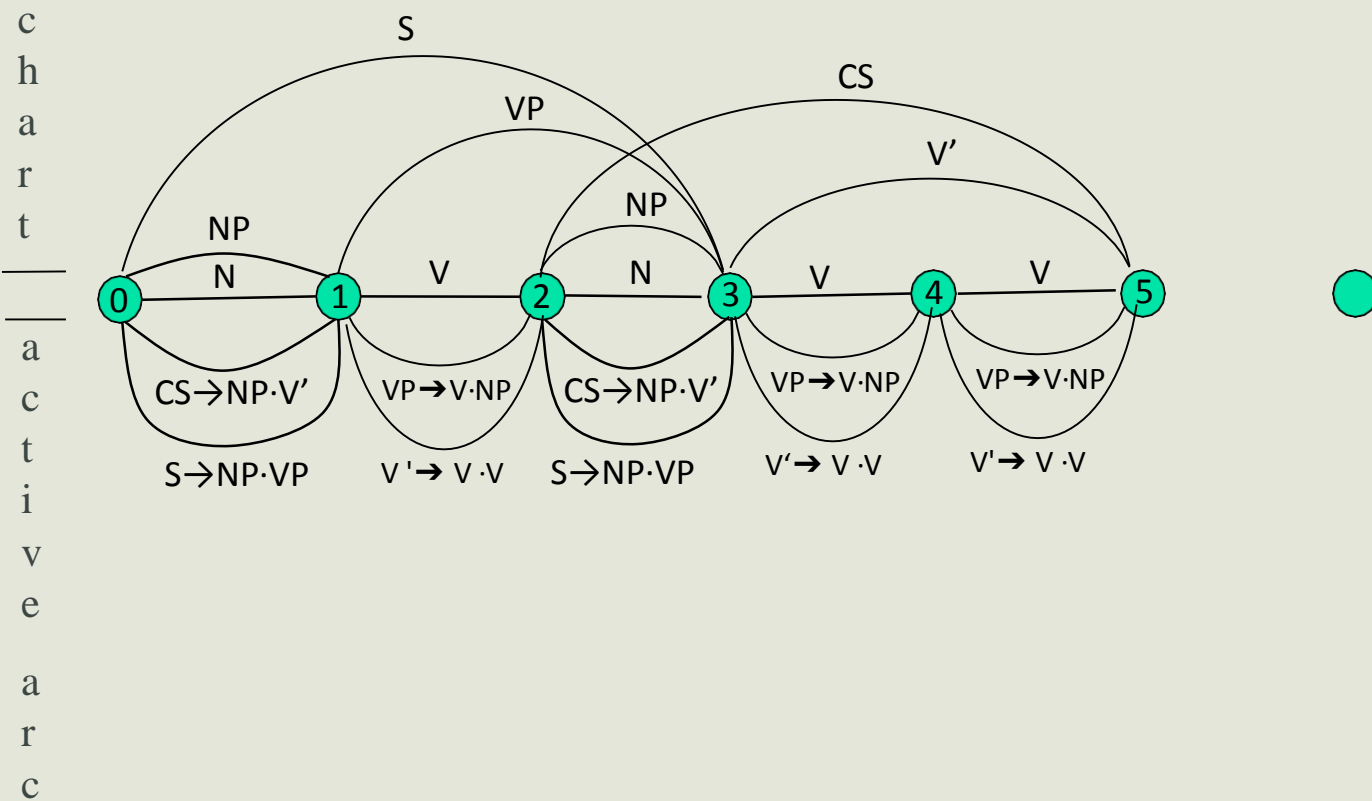
agenda

V	de
---	----

输入缓冲区

(3,4) V

Chart算法分析示例



- 文法规则：
- (1) $S \rightarrow NP VP$
 - (2) $NP \rightarrow N$
 - (3) $NP \rightarrow CS de$
 - (4) $CS \rightarrow NP V'$
 - (5) $VP \rightarrow V NP$
 - (6) $V' \rightarrow V V$

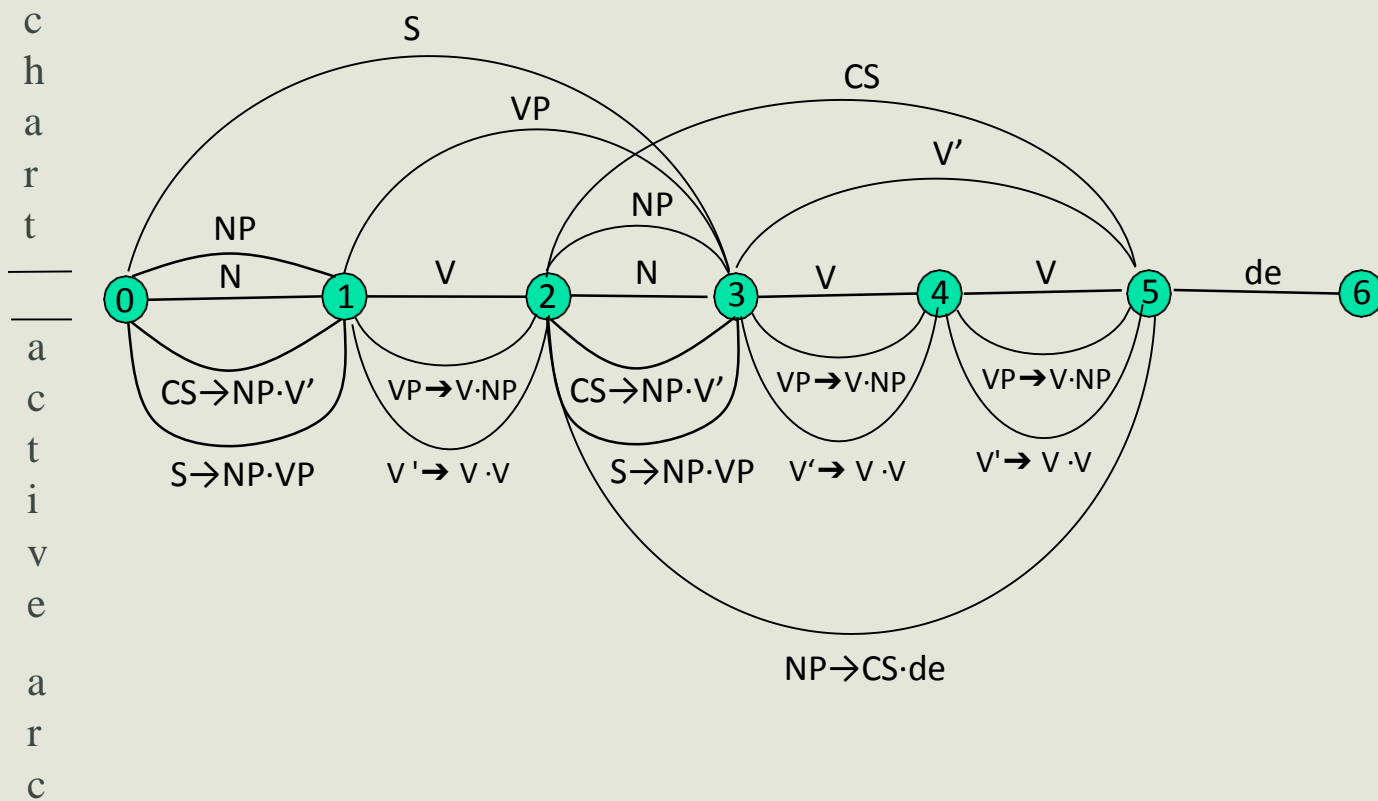
agenda

de

输入缓冲区

$((4,5)CS)$

Chart算法分析示例



文法规则：

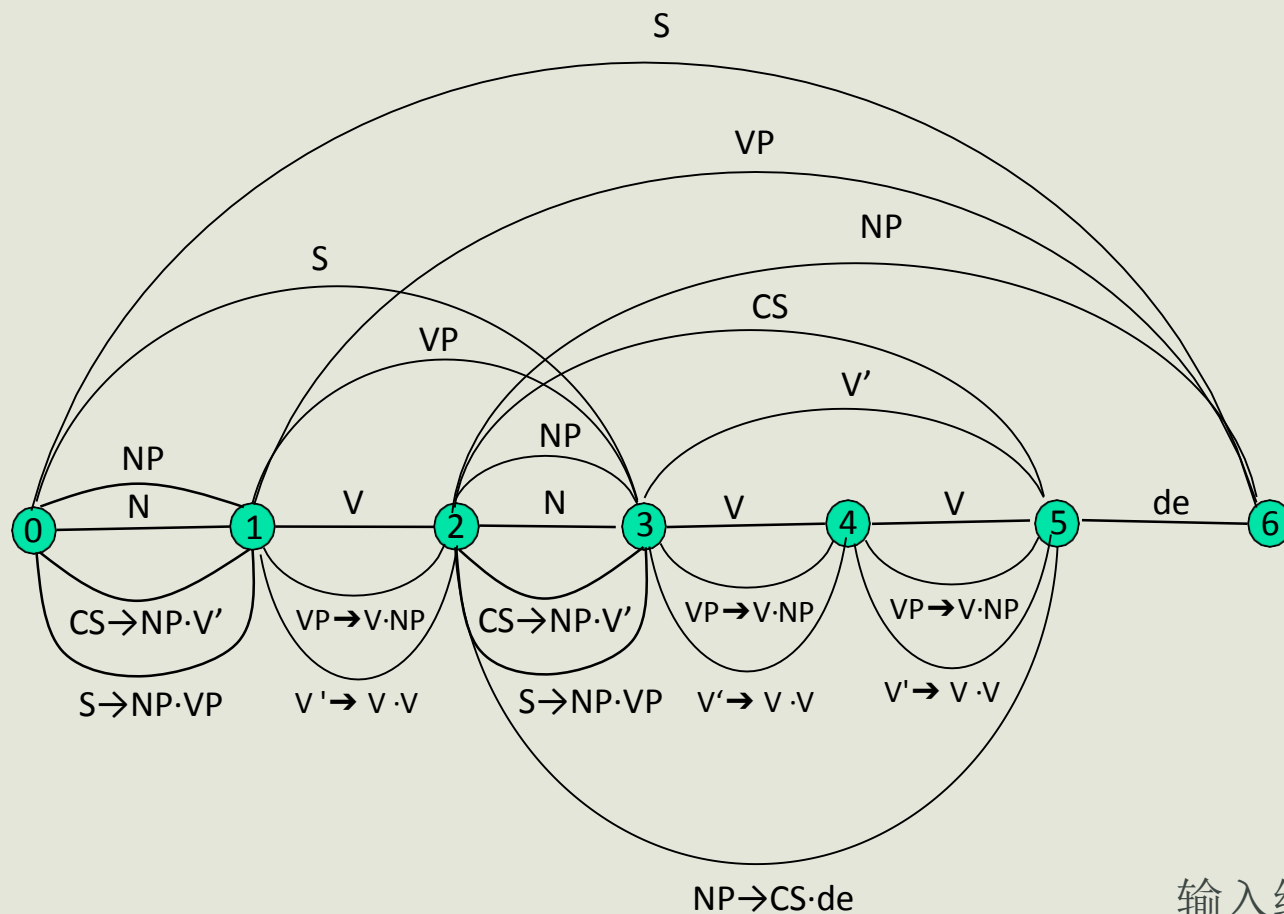
- (1) $S \rightarrow NP VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS de$
- (4) $CS \rightarrow NP V'$
- (5) $VP \rightarrow V NP$
- (6) $V' \rightarrow V V$

agenda

(5,6) de

输入缓冲区

Chart算法分析示例



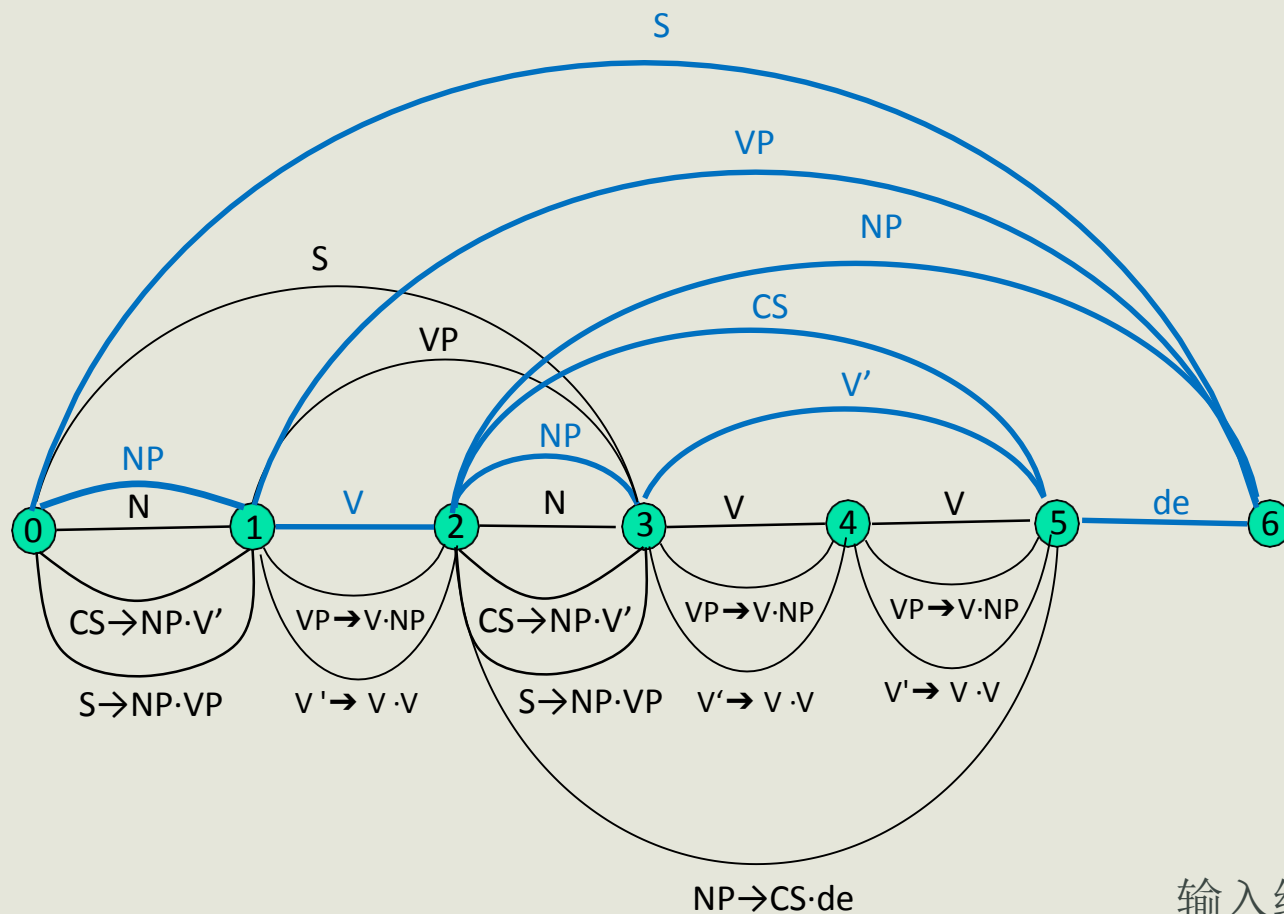
文法规则：

- (1) $S \rightarrow NP \ VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS \ de$
- (4) $CS \rightarrow NP \ V'$
- (5) $VP \rightarrow V \ NP$
- (6) $V' \rightarrow V \ V$

agenda

输入缓冲区

Chart算法分析示例



文法规则：

- (1) $S \rightarrow NP \ VP$
- (2) $NP \rightarrow N$
- (3) $NP \rightarrow CS \ de$
- (4) $CS \rightarrow NP \ V'$
- (5) $VP \rightarrow V \ NP$
- (6) $V' \rightarrow V \ V$

agenda

输入缓冲区

Chart算法分析

- 优点

- 简单、容易实现

- 弱点

- 效率不高，算法复杂度为 $O(n^3)$
 - 严重依赖句法规则质量
 - 难以区分歧义结构