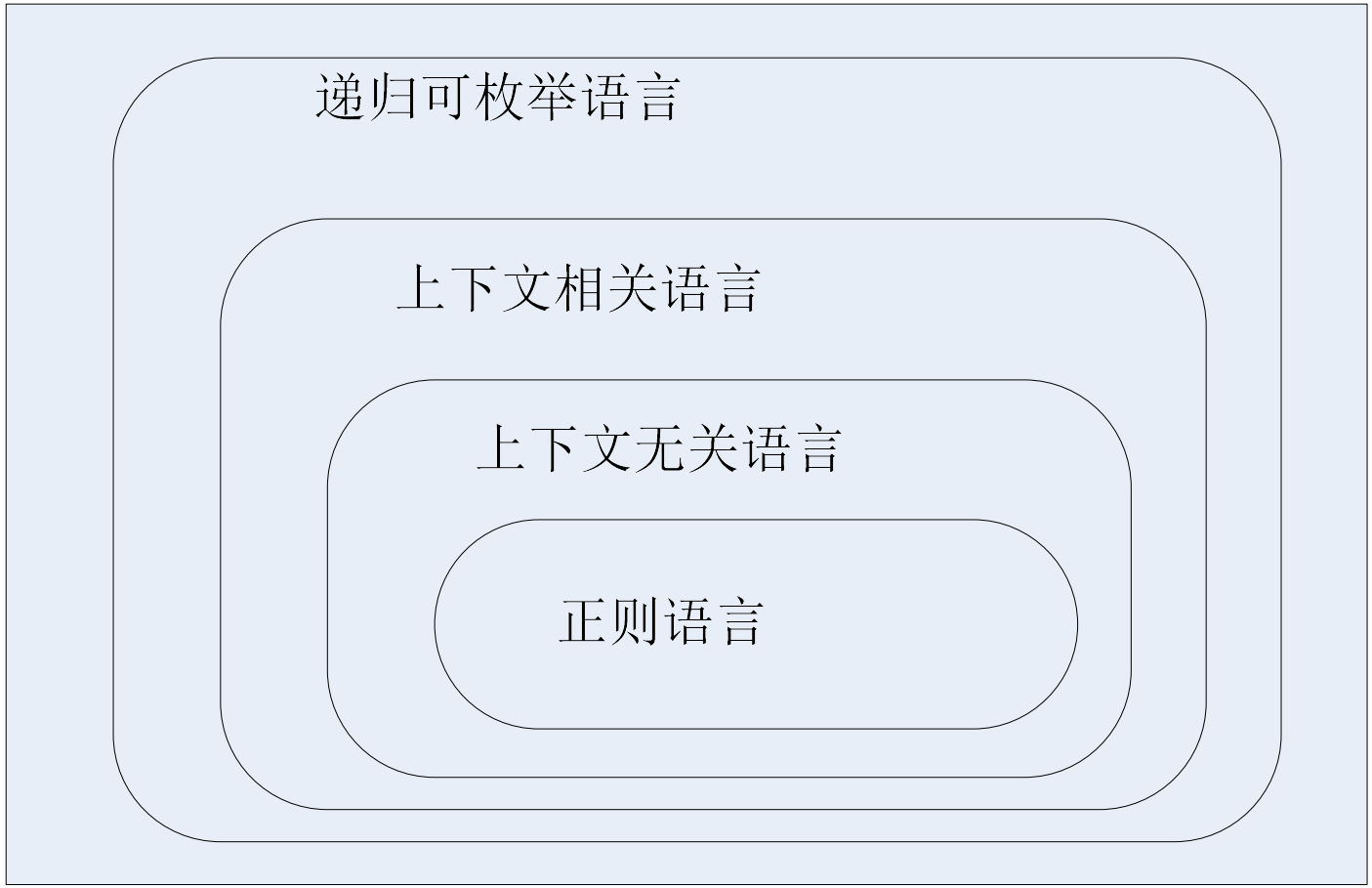
语言与复杂性(Language and Complexity)
--从形式语言体系看

王小捷
智能科学与技术中心
北京邮电大学

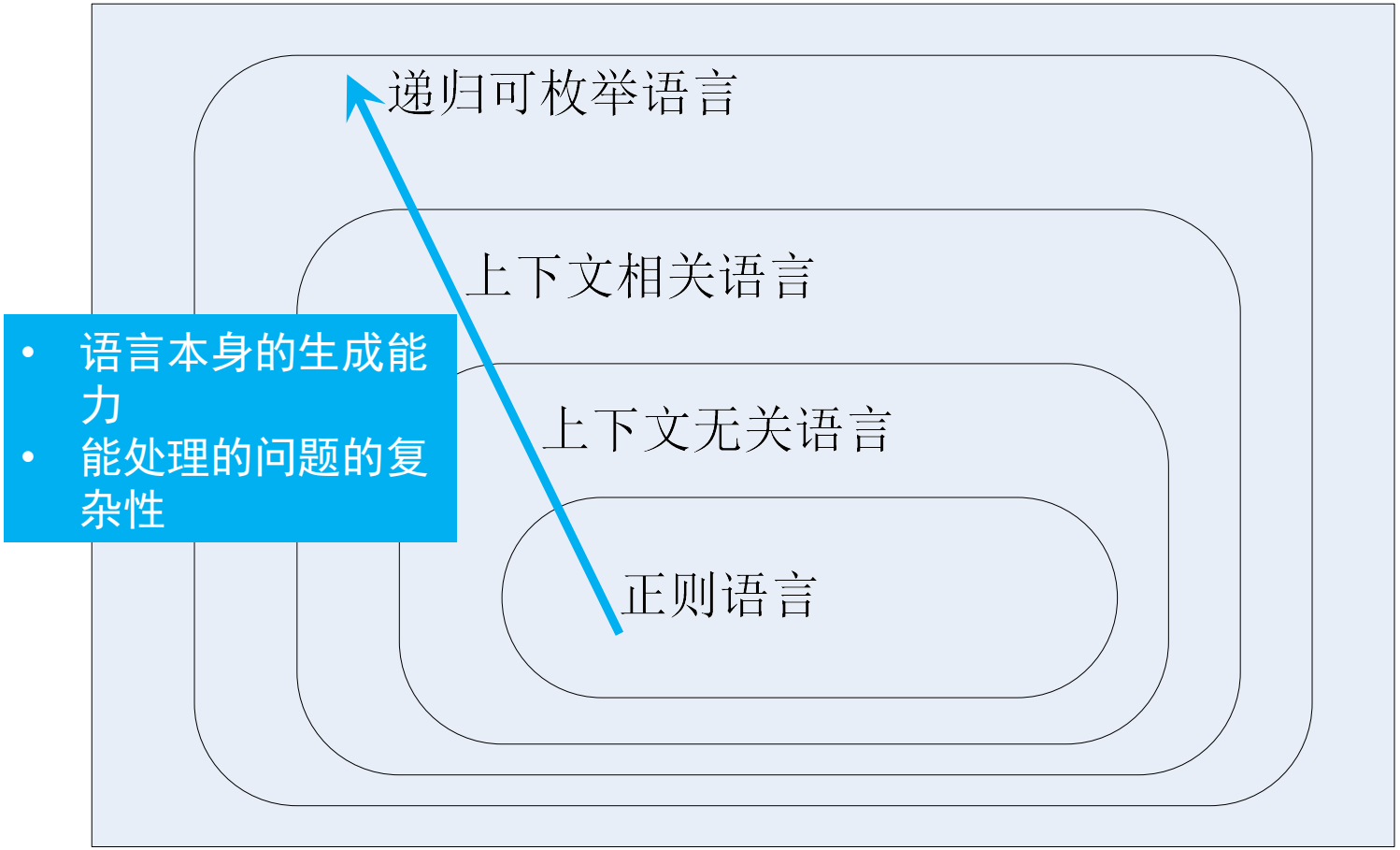
主要内容

- Chomsky 层次
- 自然语言不是正则语言(RL)
- 自然语言不是上下文无关语言(CFL)
- 自然语言是什么？

Chomsky层次

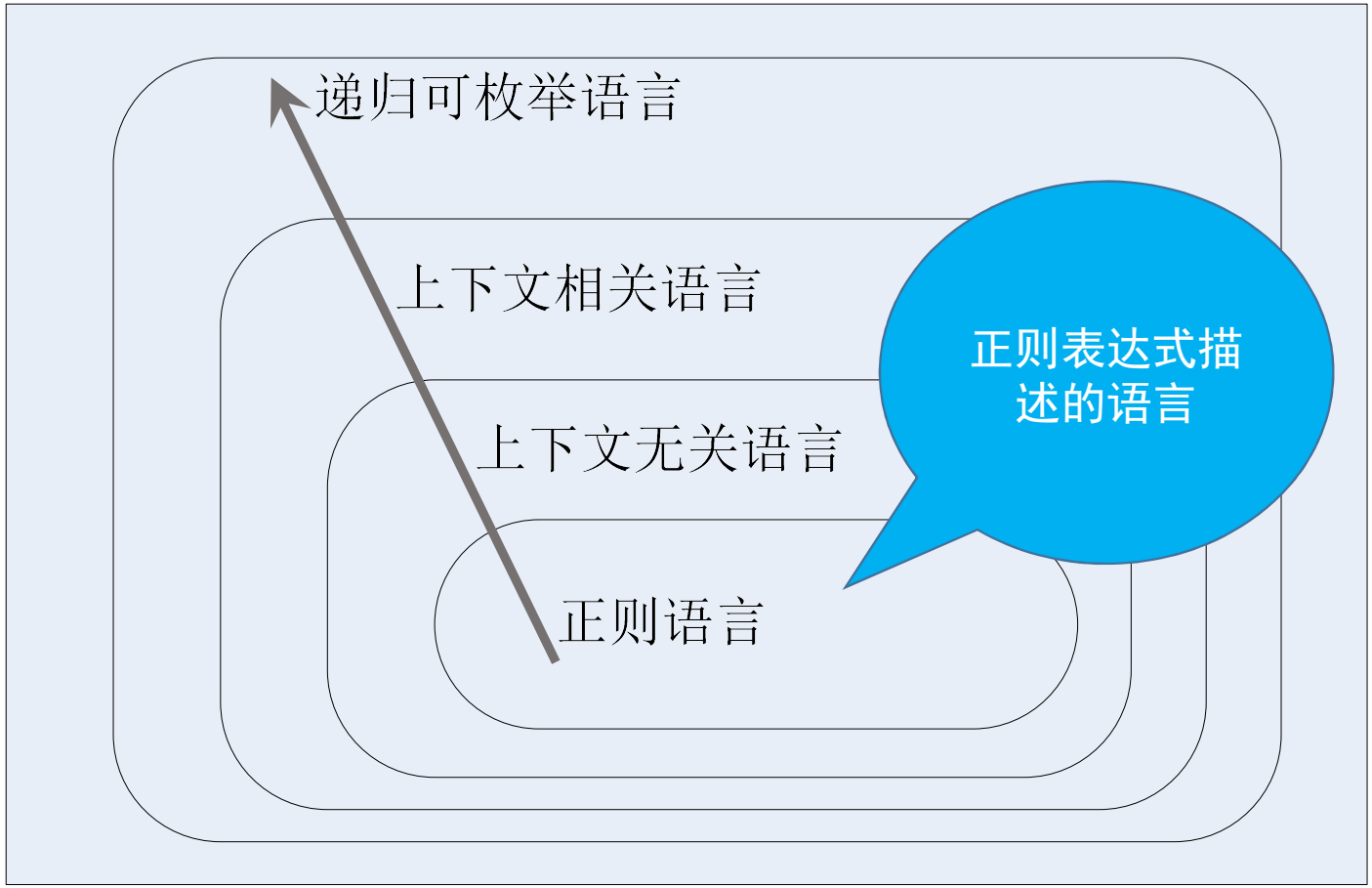


Chomsky层次

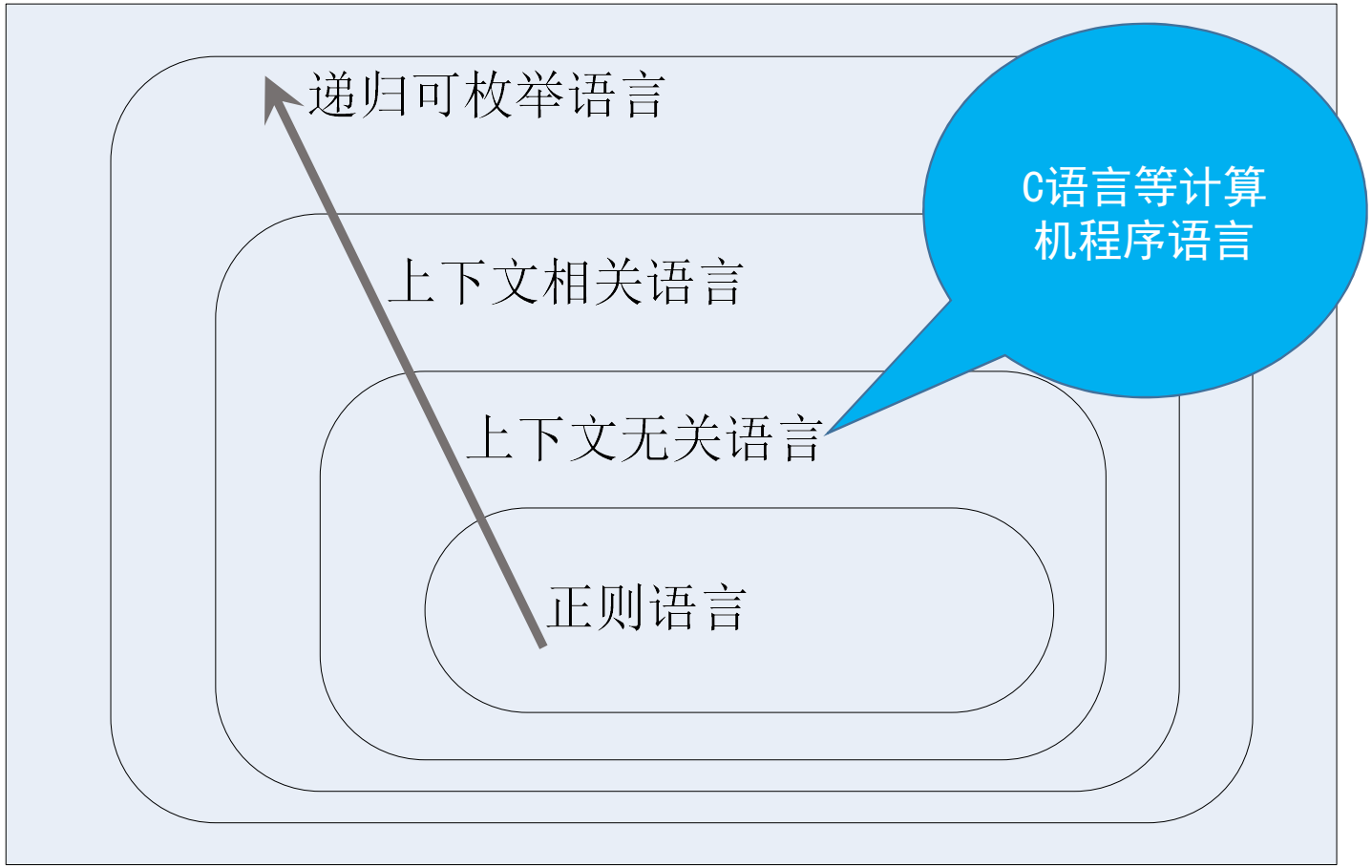


- 语言本身的生成能力
- 能处理的问题的复杂性

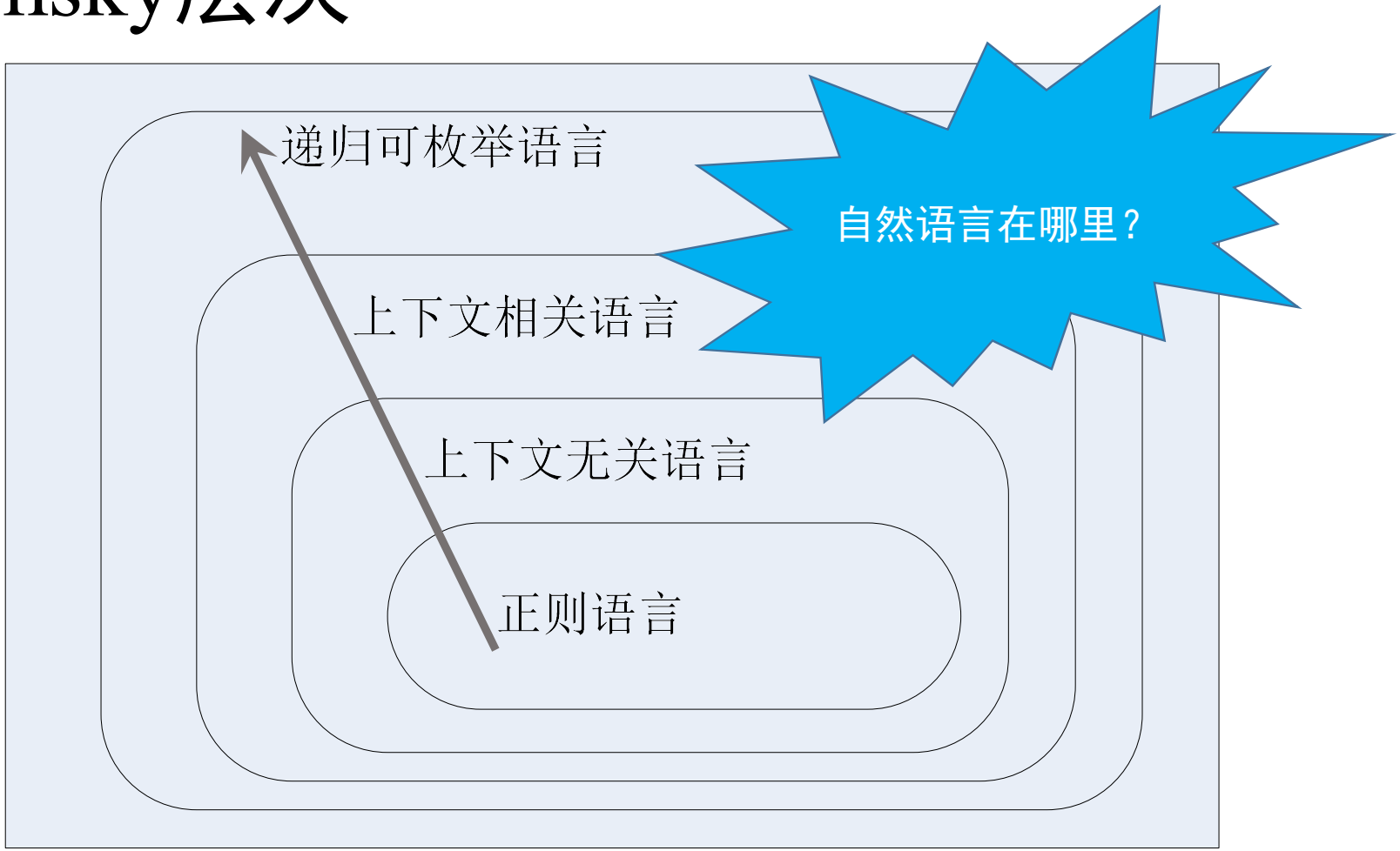
Chomsky层次



Chomsky层次



Chomsky层次



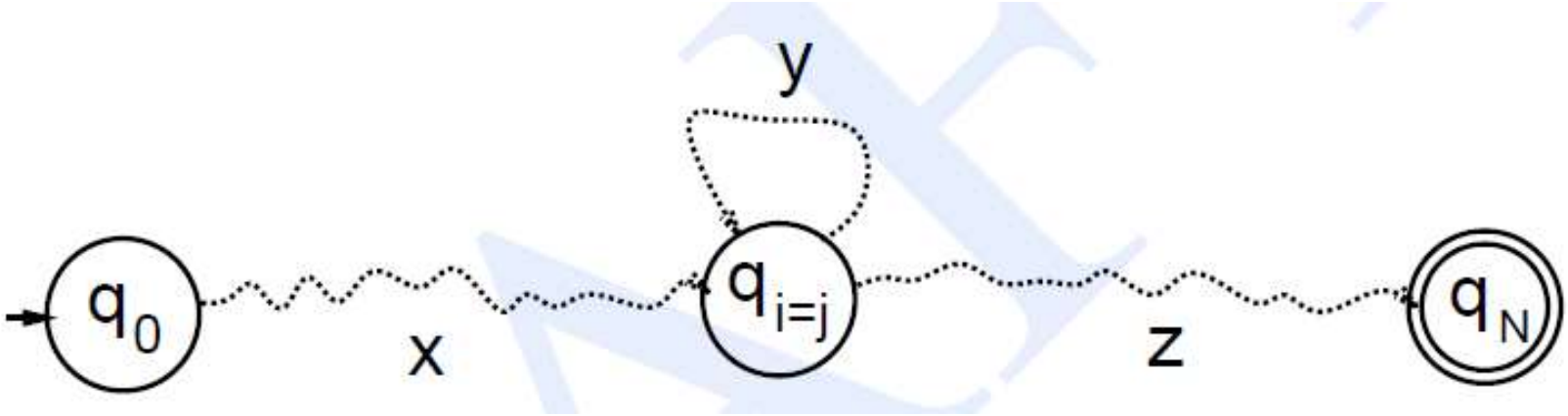
主要内容

- Chomsky 层次
- 自然语言不是正则语言(RL)
- 自然语言不是上下文无关语言(CFL)
- 自然语言是什么？

■正则语言的泵引理(Pumping Lemma)

■设L是一个无限正则语言，则存在串 $x, y,$ 和 $z,$ 满足 $y \neq \varepsilon$ 且 $xy^n z \in L(n \geq 0)$。

■即：如果L是正则语言，则L中必有形如$xy^k z$的串(y非空)。



- 泵引理给出了语言L是正则语言的必要条件，但非充分，因此不能用于判断语言L是正则语言。
- 但其逆否命题则给出了语言L不是正则语言的充分条件，即：
- **如果L中没有任何形如xy^kz的串(y非空)，则L不是正则语言。**
  - 因此，要证明语言L不是正则语言，只需要证明该语言没有任何形如xy^kz的串(y非空)。

- 证明$a^n b^n$型语言不是正则语言。
  - 只需要证明$a^n b^n$没有任何形如$xy^k z$的串
  - 分几种情况证明：
  - $y=a$ 则$x=a, z=b$, 则a至少比b多1
  - $y=b$ 则$x=a, z=b$, 则b至少比a多1
  - $y=ab$ 则有$(ab)^n$，意即味着有b在a前

- 于是，可以得到另一个判定语言L不是正则语言的方法：
- **如果语言L包含$a^n b^n$型语言，则L不是正则语言。**

■英语不是正则语言，因为英语中存在a^nb^n型串，即center-embedded结构：

■The cat [the dog [the rat [the goat licked] bit] chased] likes tuna fish

■NP NP NP NP VP VP VP VP

■进一步：

■假设：不同的自然语言具有相同的生成能力。

■因此：

■英语不是正则语言，则所有其他自然语言都不是正则语言。

主要内容

- Chomsky 层次
- 自然语言不是正则语言(RL)
- 自然语言不是上下文无关语言(CFL)
- 自然语言是什么？

■上下文无关语言的泵引理(Pumping Lemma)

■设L是一个无限上下文无关语言，则存在串 u, v, w, x和y, 满足 $v,w,x \neq \varepsilon$ 且 $uv^n wx^n y \in L(n \geq 0)$。

■即：如果L是上下文无关语言，则L中必有形如 $uv^n wx^n y$的串(v,w,x非空)。

- 同样地，上下文无关的泵引理给出了语言L是上下文无关语言的必要条件，但非充分，因此不能用于判断语言L是上下文无关语言。
- 但其逆否命题则给出了语言L不是上下文无关语言的充分条件，即：
- **如果L中没有任何形如uv^nwx^ny的串(v,w,x非空)，则L不是上下文无关语言。**
  - 因此，要证明语言L不是上下文无关语言，只需要证明该语言没有任何形如形如uv^nwx^ny的串(v,w,x非空)。

- 证明$a^n b^m c^n d^m$型语言不是上下文无关语言。
  - 只需要证明$a^n b^m c^n d^m$没有任何形如$uv^n wx^n y$的串
  - 类似地也可分情况证明
  - …

- 于是，可以得到另一个判定语言L不是上下文无关语言的方法：
- **如果语言L包含$a^n b^m c^n d^m$型语言，则L不是上下文无关语言。**

■瑞士德语不是上下文无关语言，因为其中存在$a^n b^m c^n d^m$型串，即cross-serial dependencies (跨序列依赖)结构。

■简单的例子开始：

■*mer em Hans es huus h¨alfed aastriiche.*

■we Hans the house helped paint.

■英译：we helped Hans paint the house.

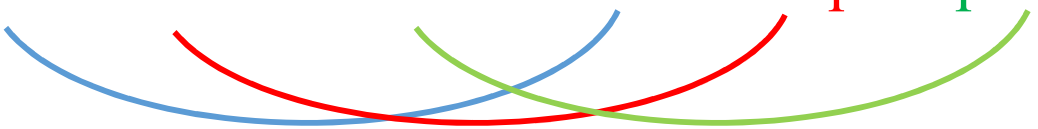
■简单的例子开始：

■*mer em Hans es huus h¨alfed aastriiche.*

■we Hans the house helped paint.

■英译：we helped Hans paint the house.

■复杂一点：We child Hans the house let helped paint.



■进一步复杂：

■一种可能➔

■We child-1 child-2 Hans the house let-1 let-2 helped paint.

■→

■We (child)* Hans the house (let)* helped paint.

■另一种可能➔

■We child Hans-1 Hans-2 the house let helped-1 helped-2 paint.

■→

■We child (Hans)* the house let (helped)* paint.

■两种可能是独立的，可进一步结合

■We (child)* (Hans)** the house (let)* (helped)** paint.

■即$a^m b^n c^m d^n$形式。

- 进一步：
  - 假设：不同的自然语言具有相同的生成能力。
- 因此：
  - 瑞士德语不是上下文无关语言，则所有其他自然语言都不是上下文无关语言。

主要内容

- Chomsky 层次
- 自然语言不是正则语言(RL)
- 自然语言不是上下文无关语言(CFL)
- 自然语言是什么？

- 从形式语言的角度：
  - 自然语言至少是上下文相关语言
  - 已有图灵机等价的语法对自然语言进行描述(HPSG\LFG等)，也就是将语言看为递归可枚举语言，采用图灵机建模
- 基于图灵机的自然语言处理

图灵机M(Q, q_0, F, Γ, b, ∑, δ)

Q	有限非空状态集合
q_0	$q_0 \in Q$，初始状态
F	$F \subseteq Q$，终止状态集
Γ	带符号表，可以出现在输入带上的所有符号的集合
b	$b \in \Gamma$，空白符号
∑	$\sum \in \Gamma \backslash \{b\}$，输入符号集
δ	移动函数$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

基于图灵机的自然语言处理—如果采用预定义的转移函数

- 设$p, q \in Q;\ X,Y \in \Gamma;\ D \in \{L,R\}$
- 转移函数$\delta : (p, X) \rightarrow (q, Y, D)$
  - 如果(q, Y, D) 唯一确定，即δ是确定性的，能处理语言的歧义(不确定性)？
  - 如果(q, Y, D) 不唯一，即δ是不确定性的，能处理语言的歧义(不确定性)？
    - 不确定性图灵机存在等价的确定性图灵机？
- 问题
  - 1)人工预定并不再变化；2)不与环境交互并随之变化

基于图灵机的自然语言处理—如果在线自主获得转移函数

- 机器自主获得转移函数的可能途径
  - 1)基于标注样例学习转移函数：监督学习
  - 样例：$(p_i, X_i) \rightarrow (q_i, Y_i, D_i)$ i=1,2,…n
  - 学习：$\delta : (p, X) \rightarrow (q, Y, D)$
    - 相关：已知$\delta : (p_i, X_i) \rightarrow (q_i, Y_i, D_i)$，那么$\delta : (p_i + \varepsilon, X_i + \varepsilon) \rightarrow ?$
  - 2)基于评价信息学习转移函数：强化学习
  - ……

- 似乎基于图灵机的自然语言处理就是能有效自主习得转移函数，即：核心在于具有学习能力，而学习的结果是获得合适的转移函数。
- 目前，机器学习已成为很多自然语言处理技术的核心，产生了很多学习算法。
- 但是，学习转移函数的算法也是图灵机！
- 是无穷递归，还是存在原初的确定性图灵机？
- 这可能已经不是图灵机本身能回答的问题了！

依存分析(Dependency Parsing)
----语言学语法

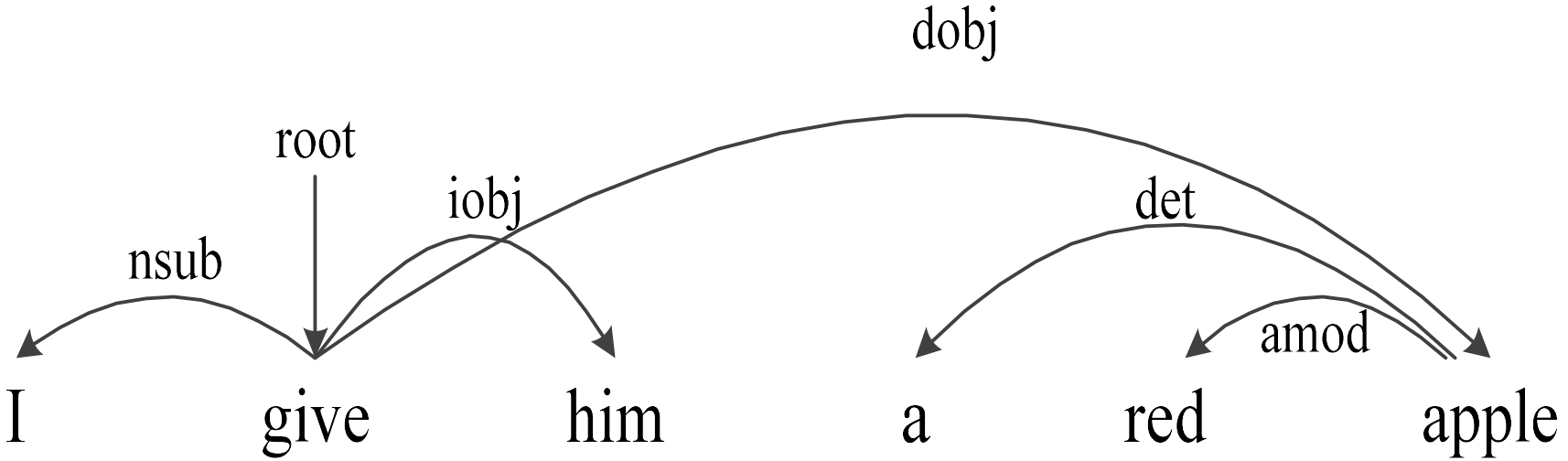
王小捷
智能科学与技术中心
北京邮电大学

- CFG：基于单元结构
- 直接建立词汇间的关联
  - 依存语法(Dependency Grammar, DG)

- 依存语法DG
  - 法国特斯尼耶尔(Lucien Tesniere)
  - 1959年《结构句法基础》
  - 基本观点：
    - 句子是一个整体，构成成分是词
    - 词间具有(依存)关系，这些(依存)关系构成了句子的框架
    - 原则上一个依存关系连接两个词，一个是支配词，一个是从属词，一个词既可以是支配词，也可以是从属词
    - 句子中只作为支配词不做从属词的词是句子的核心，其一般是动词

■基于依存语法的句子结构描述

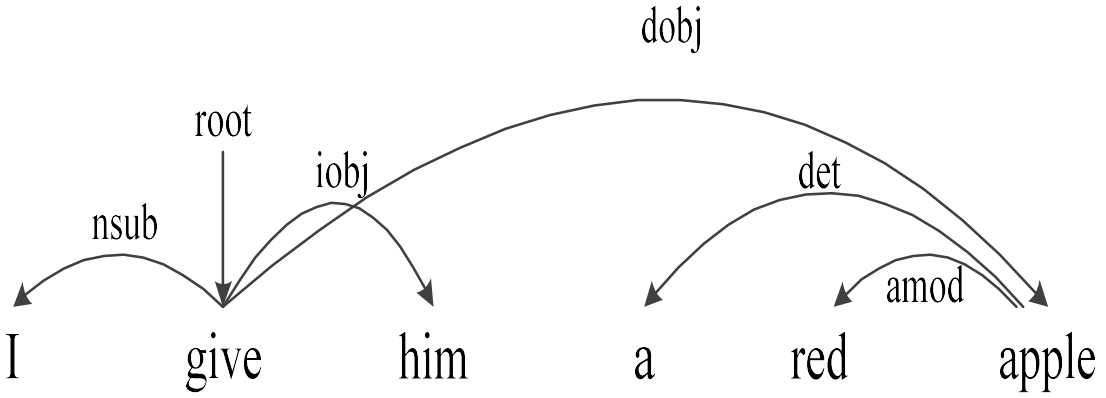
- ■词
- ■词间关系



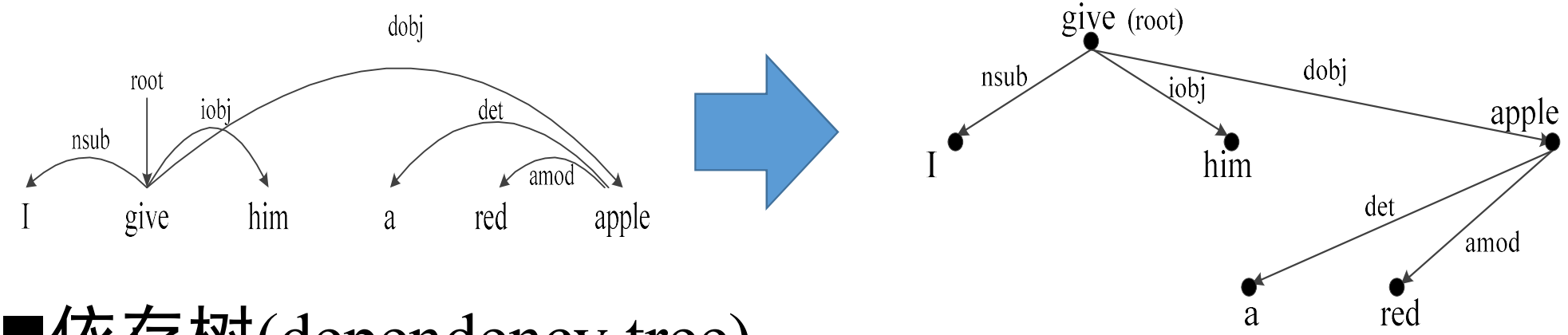
■词间关系

论元关系	说明
NSUBJ	名词性主语
DOBJ	直接宾语
IOBJ	间接宾语
CCOMP	子句内补语
名词性修饰关系	**说明**
NMOD	名词性修饰词
AMOD	形容词性修饰词
NUMMOD	数词性修饰词
DET	限定词
其他关系	**说明**
CONJ	连词
CC	并列连词

- 依存分析的作用
  - 句子主干结构
    - 主动词root=give
    - 句子主体：基于主动词的关系
    - give(I him apple)
  - 单元组块
    - A red apple
  - 词间关系
    - 三元组
    - (词 关系 词)

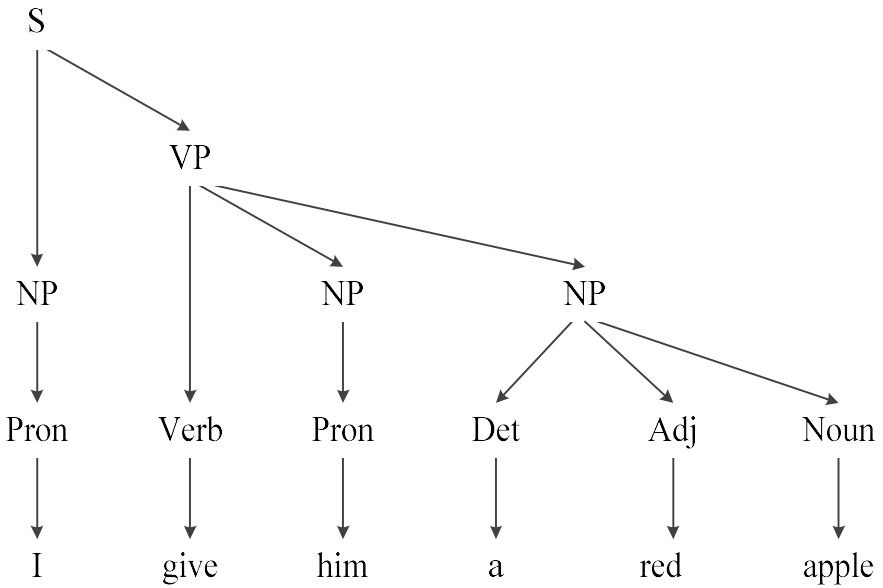


■一个依存结构是一个有向图G=(V,A)



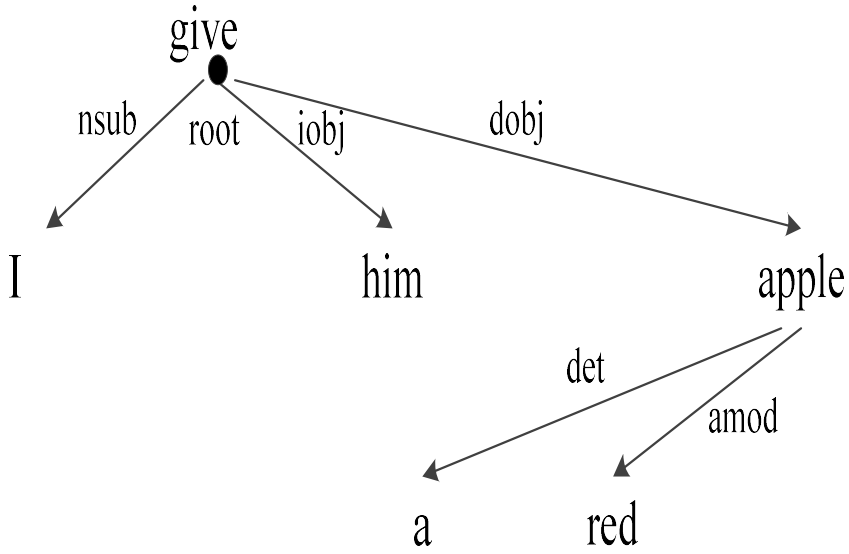
■依存树(dependency tree)
  ■一个根节点：没有输入弧
  ■其他节点有且仅有一个输入弧
  ■从根节点到其他任何节点有且仅有一条路径
■每一个子树就是一个词间关系：三元组
■每棵依存树就是一个关系集合：三元组集合

■句法树 vs 依存树



中间单元

≠

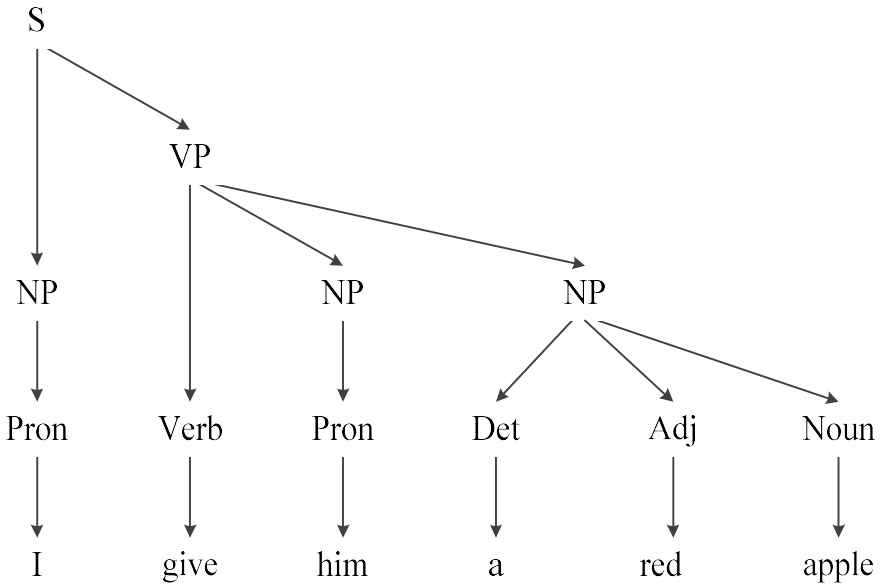


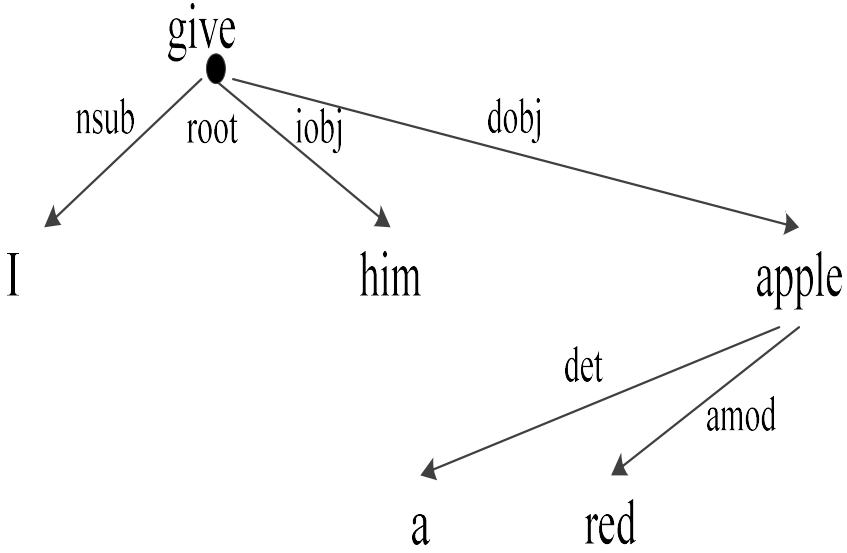
词间关系

■依存树：从句法树构建？

■基于句法树构建(无关系的)依存树

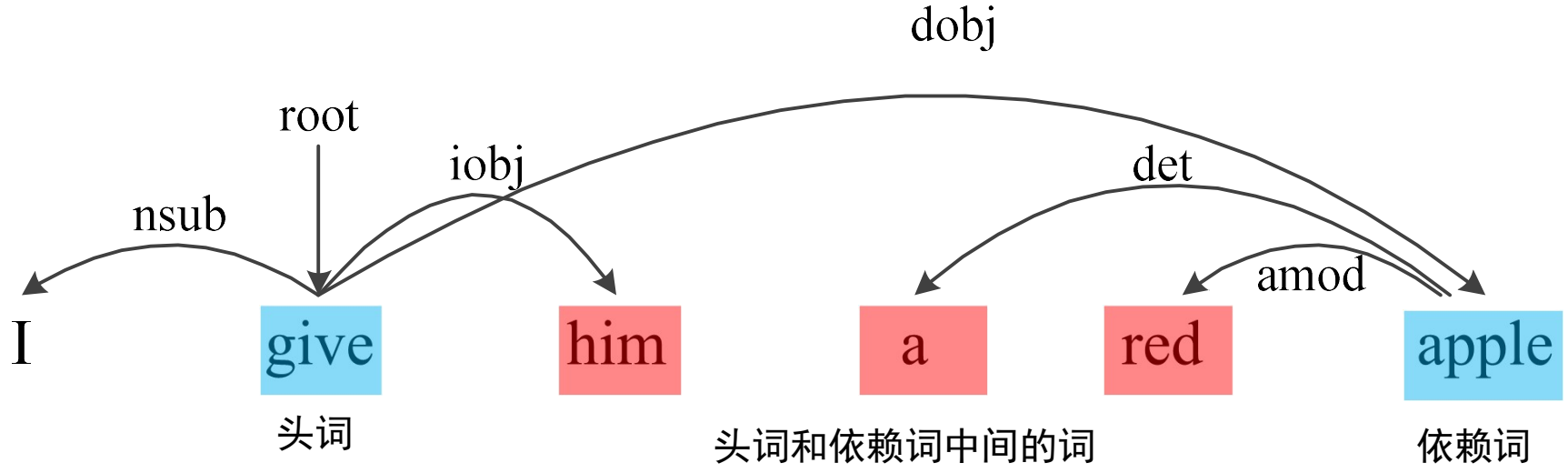
■需要结合head finding





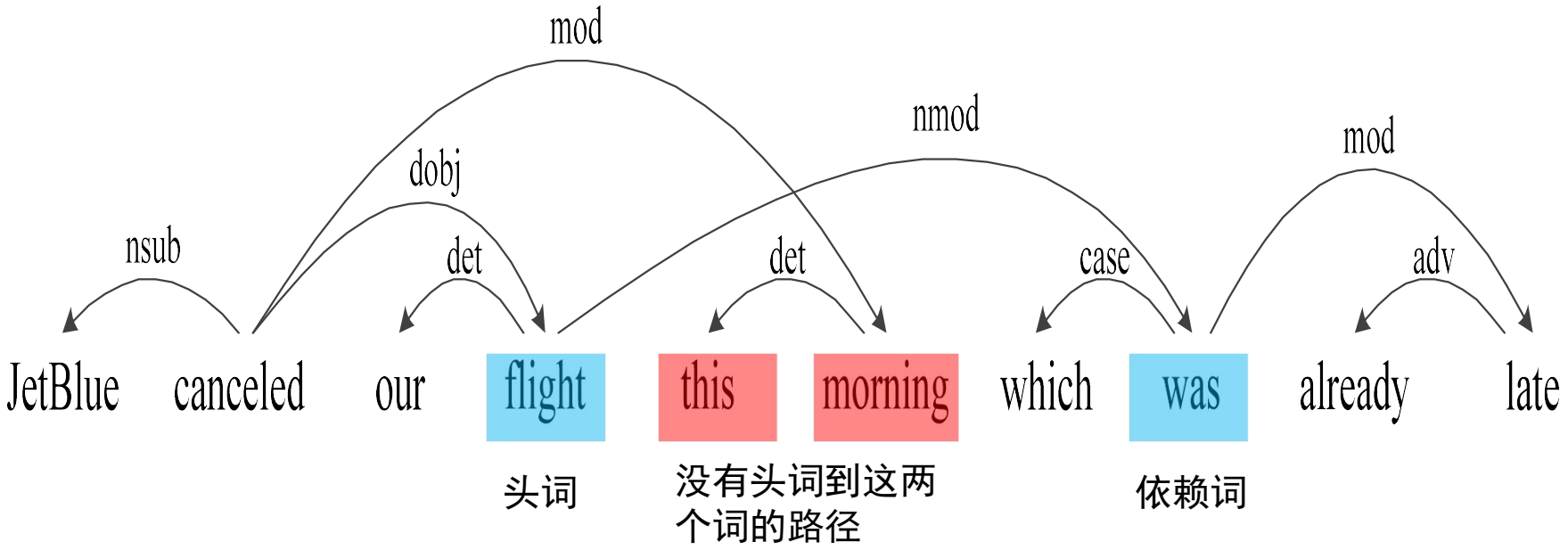
- 发展独立的依存分析方法的必要性：
  - 1）基于句法树不能得到依存树中的关系
  - 2）句法树只能产生投射性依存树，而非投射性依存广泛存在
    - 虽然非投射性带来复杂性，算法上难以处理
    - 很多有效算法只能产生投射性弧
- 因此，需要发展单独的依存分析

- 依存树的投射性(projectivity)
  - 一条弧称为是投射的：如果从头词到(其某个依赖词之间的)每一个词的路径都位于头词到该依赖词之间。



■依存树的投射性(projectivity)

■投射性在语言中不总是能得到满足



- 依存分析方法
- 基于转移（Transition-based）
  - 通过一系列移进、规约等转移动作构建一棵依存句法树，寻找最优动作序列是算法的目标。
- 基于图（Graph-based）
  - 将依存句法分析看成从完全有向图中寻找最大生成树的问题，图中的边表示两个词之间存在某种句法关系的可能性。

基于转移的依存分析

- 算法：基于状态转移的(类似于Earley)
  - 从初始状态开始，不断选择对状态的操作使得状态发生转移，最终到目标状态

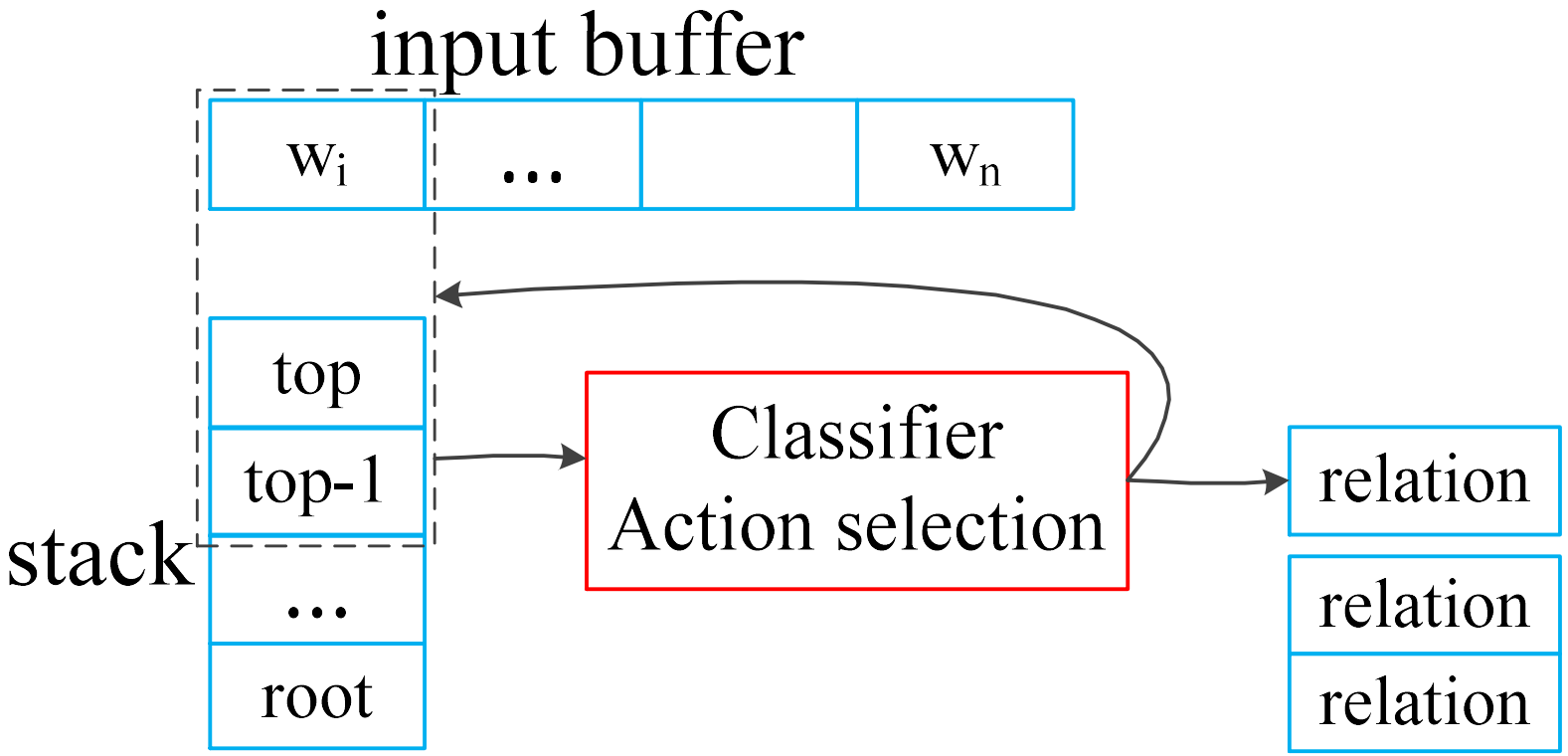
基于转移的依存分析

- 状态(configuration)
  - s={栈中内容，缓存中内容，表示依存树的关系集合}
  - 初始s_0 ={[root],[w_1,…,w_n],[]}
  - 终止s_{end}={[root],[],[依存关系集合]}

- 操作(Covington 2001, Arc-standard方法)
  - LeftArc(左规约)：得到栈最上面两个词的依存关系(下依赖上：上→下)，并移除下面那个词(除非下面那个词是ROOT)。
  - RightArc(右规约)：得到栈最上面两个词的依存关系(上依赖下：上←下)，移除上面那个词
  - Shift(移进)：从缓存中读(移)出一个词压入栈

- 操作选择
  - 由分类器(Oracle)依据当前状态选择上述操作中的一个，执行后进入新状态
- 算法复杂性：线性于句子长度，贪婪算法

■基于转移的依存分析：示意

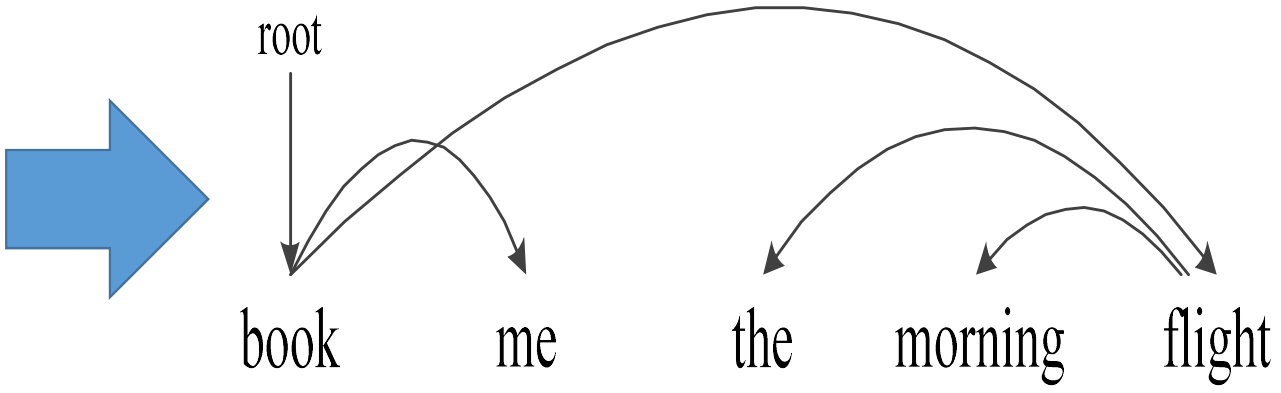


■一个例子：book me the morning flight

步骤	栈	缓存	操作	加入的关系
0	[root]	[book,me,the,morning,flight]	Shift	
1	[root,book]	[me,the,morning,flight]	Shift	
2	[root,book,me]	[the,morning,flight]	RightArc	(book→me)
3	[root,book]	[the,morning,flight]	Shift	
4	[root,book,the]	[morning,flight]	Shift	
5	[root,book,the,morning]	[flight]	Shift	
6	[root,book,the,morning,flight]	[]	LeftArc	(morning←flight)
7	[root,book,the,flight]	[]	LeftArc	(the←flight)
8	[root,book,flight]	[]	RightArc	(book→flight)
9	[root,book]	[]	RightArc	(root→book)
10	[root]	[]	Done	

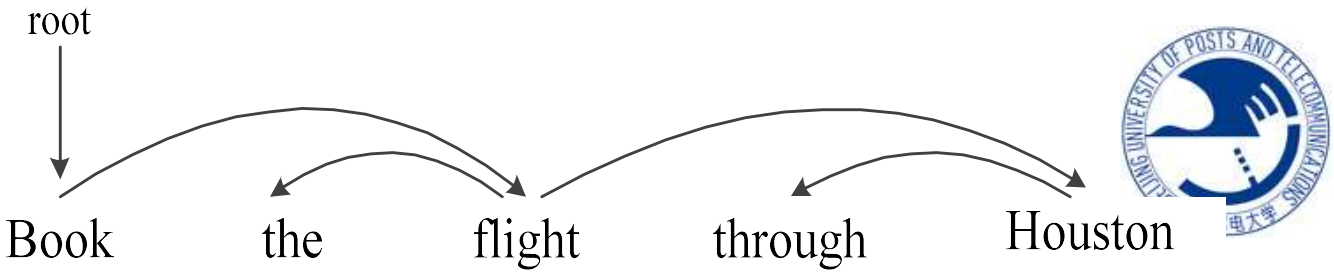
- 得到的关系集：
  - book→me
  - morning←flight
  - the←flight
  - book→flight
  - root→book

- 得到的关系集：
  - book→me
  - morning←flight
  - the←flight
  - book→flight
  - root→book



- 说明：
  - 每一个操作由分类器来决策，由算法的贪婪本质可知，分类器决策错误导致获得错的错误结果
  - 每一次可进行的操作可能不止一个，因此，可能有不同的路径，不同的路径可能导致相同的结果，或者不同的结果可能导致等价的剖析
  - 上述依存关系没有标签，为加上标签，可以将LeftArc 和 RightArc 与 标 签 进 行 组 合 ， 例 如 (LeftArc+nsub),则总操作数=2*标签数+1

- 分类器构建：监督方法
- 构建训练数据：(状态i，动作i)
- 可基于已有的依存树来构建：
  - 1、选择LeftArc：如果当前状态能产生一个正确的依存关系；
  - 2、选择RightArc：如果当前状态能产生一个正确的依存关系，且栈顶的所有依存关系都已经指派了(至少一个)；
  - 3、其他状态选择Shift。



■例子：book the flight through Houston，已有依存树如上所示

步骤	栈	缓冲	操作
0	[root]	[book,the,flight,through,Houston]	Shift
1	[root,book]	[the,flight,through,Houston]	Shift
2	[root,book,the]	[flight,through,Houston]	Shift
3	[root,book, the,flight]	[through,Houston]	LeftArc
4	[root,book,flight]	[through,Houston]	Shift
5	[root,book,flight,through]	[Houston]	Shift
6	[root,book,flight,through,Houston]	[]	LeftArc
7	[root,book,flight,Houston]	[]	RightArc
8	[root,book,flight]	[]	RightArc
9	[root,book]	[]	RightArc
10	[root]	[]	Done

- 分类器构建
  - 特征
    - 词、词性、…
    - 多词组合、…
  - 分类器:SVM
- 深度学习方法
  - Chen and Manning 2014首次
  - 无需手工构建特征，有效缓解数据稀疏问题

■Arc-standard方法的问题1)

■分析句子： book the flight through Houston

步骤	栈内	输入词串	操作	加入的关系
0	[root]	[book the flight through Houston]	Shift	
1	[root,book]	[the flight through Houston]	Shift	
2	[root,book,the]	[flight through Houston]	Shift	
3	[root,book,the,flight]	[through Houston]	LeftArc	(the←flight)
4	[root,book,flight]	[through Houston]	Shift	
5	[root,book,flight,through]	[Houston]	Shift	
6	[root,book,flight,through,Houston]	[]	LeftArc	(through←Houston)
7	[root,book,flight,Houston]	[]	RightArc	(flight→Houston)
8	[root,book,flight]	[]	RightArc	(book→flight)
9	[root,book]	[]	RightArc	(root→book)
10	[root]	[]	Done	

■注意： [root,book,flight]

- 操作(Arc-eager方法)
  - LeftArc(左规约)：得到栈最上面的词和缓存最前面的词之间的依存关系(缓存词→栈)，并移除栈最上面的词(root除外)。
  - RightArc(右规约)：得到栈最上面的词和缓存最前面的词之间的依存关系 (缓存词←栈)，将缓存最前面的词压入栈
  - Shift(移进)：从缓存中读(移)出一个词压入栈
  - Reduce(规约)：移除栈最上面的词

■Arc-eager方法分析句子： book the flight through Houston

步骤	栈内	输入词串	操作	加入的关系
0	[root]	[book the flight through Houston]	RightArc	(root→book)
1	[root,book]	[the flight through Houston]	Shift	
2	[root,book,the]	[flight through Houston]	LeftArc	(the←flight)
3	[root,book]	[flight through Houston]	RightArc	(book→flight)
4	[root,book,flight]	[through Houston]	Shift	
5	[root,book,flight,through]	[Houston]	LeftArc	(through←Houston)
6	[root,book,flight]	[Houston]	RightArc	(flight→Houston)
7	[root,book,flight,Houston]	[]	Reduce	
8	[root,book,flight]	[]	Reduce	
9	[root,book]	[]	Reduce	
10	[root]	[]	Done	

■所有关系都是首次出现就得到分析

- 转移方法的问题2)
- 贪婪决策，前面的错误不可挽回
- 加入beam search来缓解
- 每次不是保留唯一的状态序列，而是保持多个(beam宽度决定的N)得分排名靠前的状态序列，每个序列均进行推进，当生成多个时删除分数低的，保留N个直到结束状态
- 因此：对状态要有一个得分评估？
- 分类器每个操作的得分作为此时状态的得分，累积得到状态序列的得分

谢谢！