依存句法分析

主讲人: 张栋

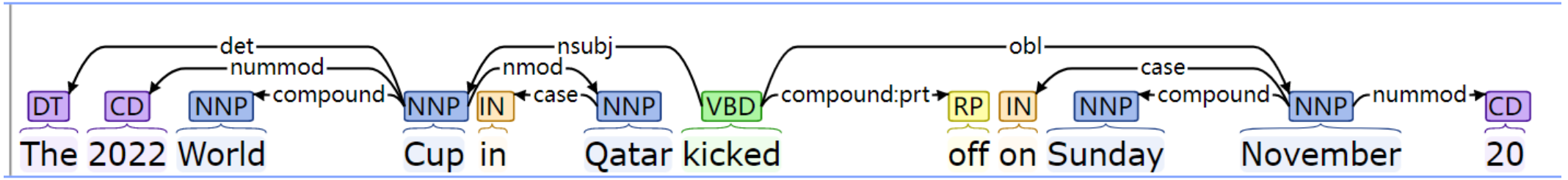
依存句法分析

又称依存关系分析

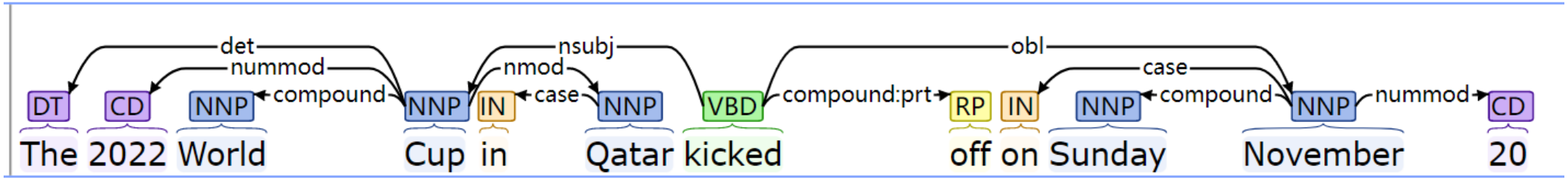
- 识别出句子中的**词语与词语之间的相互依存关系**
- 描述词语之间在句法上的搭配关系，这种搭配关系是和语义相关联的

• 依存句法

- 认为“谓语”中的动词是一个句子的中心，其他成分与动词直接或间接地产生联系
- 依存语法存在一个共同的基本假设：句法结构本质上包含词和词之间的依存（修饰）关系。一个依存关系连接两个词，分别是核心词（head）和依存词（dependent）。依存关系可以细分为不同的类型，表示两个词之间的具体句法关系。

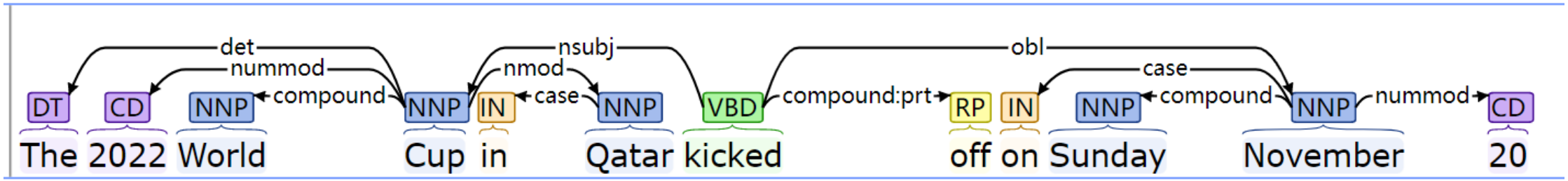


- 依存关系
  - 依存关系由**核心词**（head）与**依存词**（dependent）表示，每个核心词对应其成分的中心
  - 两大类关系
    - **从句关系**（clausal relations）
    - **修饰语关系**（modifier relations）

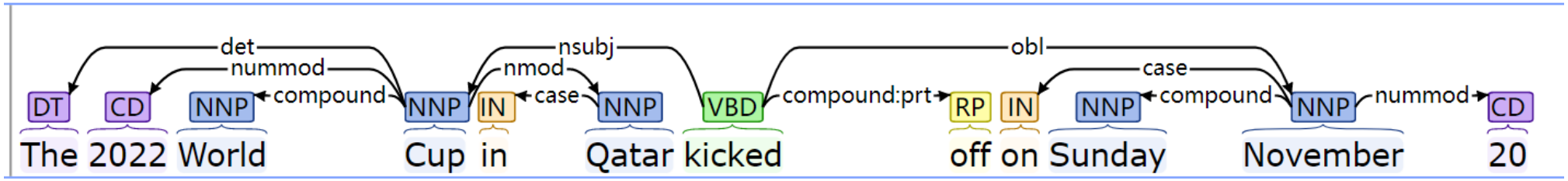


Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

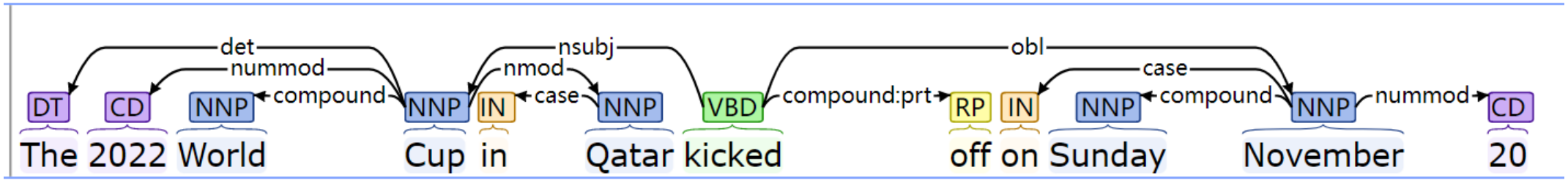
Universal Dependency 依存关系举例



- 依存形式（Dependency Formalisms）
  - **依存树**（dependency tree）是这样一种有向图
    - 有单个指定的根节点，该根节点没有传入的弧线（例如，例句中的「kicked」就是根节点）
    - 除了根节点之外，每个节点都**有且仅有**一条传入的弧线



- 投射性（Projectivity）
  - 从中心词（head）到关联词（dependent）存在一条路径，就说这条路径是投射的（projective）
  - 如果依存树里面的所有的弧线都是投射的，则这个依存树就是投射的
  - 一个依存树是否具有投射性，可以由依存树是否存在交叉的弧线来判断



- 依存树库（Dependency Treebanks）
  - 通过人工标注
  - 通过成分树库直接转换
    - 识别结构中所有的中心词-依存关系
    - 为这些关系识别出正确的依存关系

• 依存分析器的性能评价

- 无标记依存正确率（unlabeled attachment score，UAS）
- 带标记依存正确率（labeled attachment score， LAS）
- 依存正确率（dependency accuracy，DA）
- 根正确率（root accuracy，RA）
- 完全匹配率（complete match，CM）

- **依存分析器的性能评价**

无标记依存正确率（*unlabeled attachment score*，*UAS*）：

测试集中找到其正确支配词的词（包括没有标注支配词的根结点）所占总词数的百分比。

• 依存分析器的性能评价

- 带标记依存正确率（*labeled attachment score*， *LAS*）
  - 测试集中找到其正确支配词的词，并且依存关系类型也标注正确的词（包括没有标注支配词的根结点）占总词数的百分比。

• 依存分析器的性能评价

- 依存正确率（dependency accuracy，DA）
  - 测试集中找到正确支配词非根结点词占所有非根结点词总数的百分比。

• 依存分析器的性能评价

- 根正确率（root accuracy，RA）
  - 测试集中正确根结点的个数与句子个数的百分比。
  - 测试集中找到正确根结点的句子数所占句子总数的百分比。

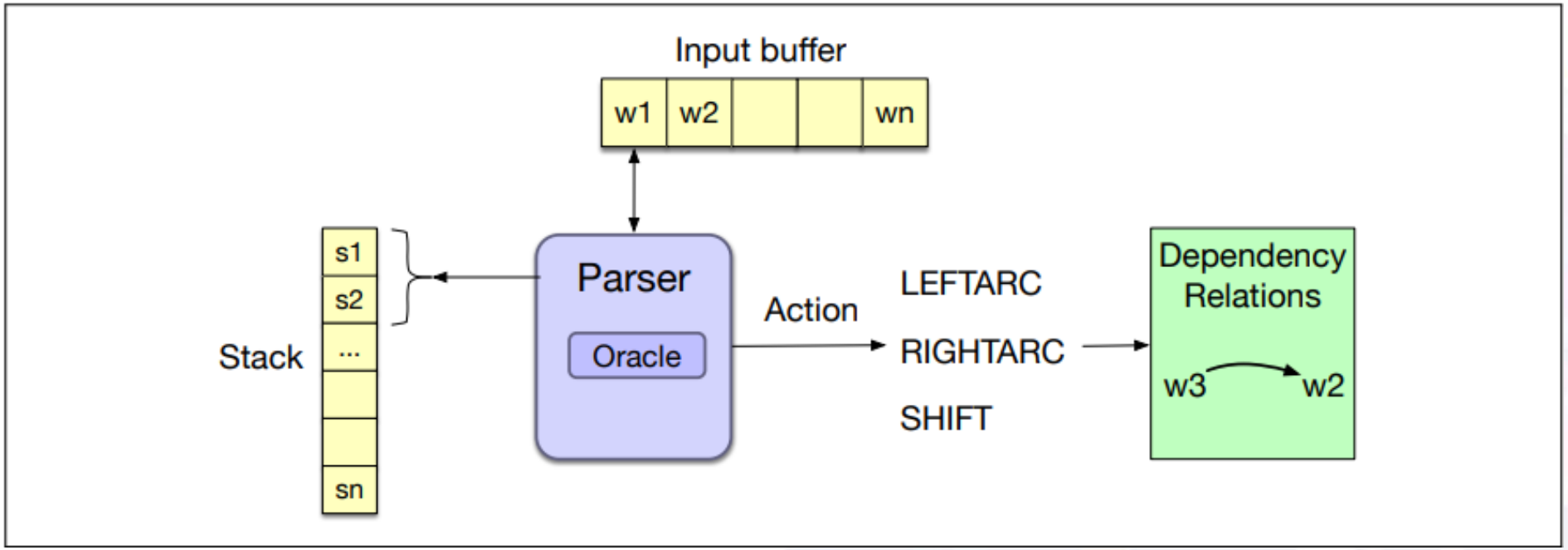
依存分析器的性能评价

- 完全匹配率（complete match，CM）
  - 测试集中无标记依存结构完全正确的句子占句子总数的百分比。

• 基于转移的依存分析

(Transition-Based Dependency Parsing)

- 基于堆栈的方法，称为**移进归约**(shift-reduce)分析
- 三个组成部分
  - 一个堆栈 (stack), 用于构建依存树, 初始为空
  - 一个缓冲区 (buffer), 用于存放待分析的单词列表, 初始为组成待分析句子的单词列表
  - 一个分析器 (parser), 该分析器根据预言机(Oracle)预测的动作 (action)，执行相应的操作



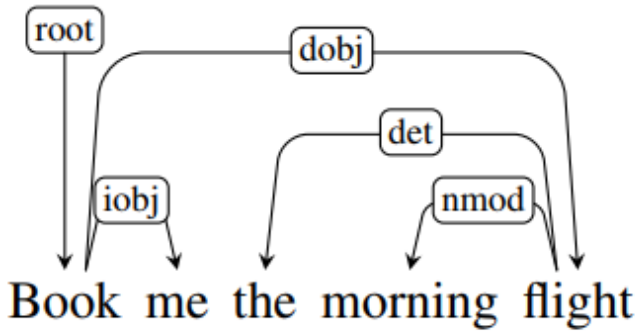
- 分析器从左至右浏览句子，将单词从缓冲区移进堆栈
- 根据当前状态 (state)，预言机预测动作，分析器执行动作

三种动作

- LEFTARC
  - 对栈顶单词和其紧邻单词建立（中心词-关联词）关系；删除紧邻词
- RIGHTARC
  - 对紧邻单词和栈顶单词建立（中心词-关联词）关系；删除栈顶词
- SHIFT
  - 从缓存头部删除单词并将其压栈

function DEPENDENCYPARSE(*words*) **returns** dependency tree

state ← {[root], [*words*], [] } ; initial configuration
while *state* **not final**
t ← ORACLE(*state*) ; choose a transition operator to apply
state ← APPLY(*t*, *state*) ; apply it, creating a new state
return *state*



Step	Stack	Word List	Action	Relation Added
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	

谢谢